НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено

Завідувач кафедри

_____ О.В. Коваль
(підпис) (ініціали, прізвище)

“___” _____2019р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки 6.050103 “Програмна інженерія”

на тему «Система поліпшення проведення занять (Web – додаток)»

Виконав: студент 4 курсу, групи ТІ-51

Єрьоменко Олександр Сергійович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник доцент, к.т.н. Ходаківський О. В. _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент_____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший , бакалаврський

Напрям підготовки 6.050103 “Програмна інженерія”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2019р.

ЗАВДАННЯ
на дипломну роботу студенту

_____
(прізвище, ім’я, по батькові)

1. Тема роботи ___«Система поліпшення проведення занять (Web – додаток)»___
керівник роботи _____
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”__”_____ 2019р. № __

2. Строк подання студентом роботи _____
3. Вихідні дані до роботи персональний комп’ютер під керуванням операційної системи Microsoft Windows, мова програмування TypeScript.
4.Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити) вивчити проблему взаємодії викладачів та студентів, проаналізувати існуючі програмні рішення проблеми, обґрунтувати обрані програмні інструменти, розробити систему поліпшення проведення занять, розробити інтерфейс доступу до системи у вигляді веб-додатку, зробити висновки проведеної роботи.
5. Перелік ілюстративного матеріалу
1. Мета роботи. 2. Завдання роботи. 3. Опис функцій програмного продукту. 4. Опис функцій програмного продукту. 5.Функціональність системи. Діаграма прецедентів. 6. Використані програмні засоби. 7. Інтерфейс авторизації та реєстрації. 8. Інтерфейс перегляду поточної лекції. 9. Інтерфейс перегляду списку предметів. 10. Висновки до роботи.
6. Дата видачі завдання ”___”_____2019 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Вивчення та аналіз задачі		
2	Розробка архітектури та загальної структури системи		
3.	Розробка структур окремих підсистем		
4.	Програмна реалізація системи		
5.	Оформлення пояснювальної записки		
6.	Захист програмного продукту		
7.	Передзахист		
8.	Захист		

Студент _____ _____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Мета роботи – розробити систему для поліпшення проведення занять, яка б допомогла викладачам та студентам взаємодіяти більш зручним способом, та реалізувати програмний продукт у вигляді веб-застосунку. Для розробки веб-застосунку обрано фреймворк Angular, який призначений для створення односторінкових веб-додатків та дозволяє розробляти їх з використанням мови TypeScript. Доступ до такого застосунку можна отримати через браузер.

Записка містить 67 сторінок, 22 рисунка та 10 посилань.

Ключові слова: ПРОВЕДЕННЯ ЗАНЯТЬ, КОММУНІКАЦІЯ, TYPESCRIPT, ANGULAR, WEB-ДОДАТОК, ОДНОСТОРІНКОВИЙ ЗАСТОСУНОК, INTERNET.

ABSTRACT

The purpose of the work is to develop a system for improving the conduct of classes, which would help teachers and students to interact in a more convenient way, and implement a software product as a web application. The Angular Framework, which is designed to create single-page web applications and allows you to develop them using the TypeScript language, was chosen to develop the web application. Access to such an application can be obtained through a browser.

The note contains 67 pages, 22 figures and 10 links.

Keywords: CONDUCTING CLASSES, COMMUNICATION, TYPESCRIPT, ANGULAR, WEB APPLICATION, SINGLE-PAGE APPLICATION, INTERNET.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

HTTP – HyperText Transfer Protocol

SPA – Single-page application

SSA – Server-side application

DOM – Document Object Model

AJAX – Asynchronous Javascript and XML

JSON – JavaScript Object Notation

MVC – Model-View-Controller

API – Application programming interface

REST – Representational State Transfer

HTML – HyperText Markup Language

CSS – Cascading Style Sheets

URL – Uniform Resource Locator

IDE – Integrated Development Environment

CLI – Command Line Interface

ВСТУП

Навчальний процес важлива частина особи, що живе у сучасному суспільстві. Люди відвідують школи, курси, університети протягом великого проміжку свого життя. І якість цього процесу безпосередньо впливає на подальші успіхи людини у своєму житті.

У наш час, у еру технологій та інформаційних систем, автоматизування навчального процесу може значно підвисити його ефективність. Студенти та учні отримують великий об’єм інформації, яку потрібно систематизувати та згрупувати для легшого розуміння матеріалу.

Взаємодія між викладачем та учнем один з головних факторів ефективного навчання. Адже простого викладання матеріалу без зворотного зв’язку з боку студентів не достатньо для успішного сприйняття матеріалу студентами чи учнями, які присутні на лекції. Викладачу необхідно взаємодіяти зі студентами, щоб отримати для себе оцінку якості викладання свого предмету. У свою чергу, студентам потрібно мати змогу задати питання викладачу у випадку, коли у темі можуть бути незрозумілі моменти.

Отже, була поставлена задача розробити систему, яка б допомогла покращити взаємодію між викладачами та студентами під час проведення занять у аудиторіях, класах або ж у віддаленому режими. Адже під час заняття чи лекції не завжди є можливість задати питання викладачу, яке цікавить студентів і відноситься до теми заняття. А викладач у свою чергу не завжди має змогу відповісти кожному студенту, адже час заняття обмежений.

Тому система повинна надавати можливість студенту задати будь-яке питання до лекції в режимі онлайн, а викладачу відповісти на нього у зручний для нього час під час заняття або ж після нього.

Для доступу до системи було вирішено розробити веб-додаток, який би дозволив мати доступ до лекцій з будь якого пристрою, що має браузер та доступ до мережі Інтернет.

1. ЗАДАЧІ СИСТЕМИ ПОЛІПШЕННЯ ПРОВЕДЕННЯ ЗАНЯТЬ

Навчальний процес – це невід'ємна складова сучасної людини. Кожна особа проходить цей шлях протягом багатьох років, навчаючись у різних закладах. Спочатку це школа, де учні набувають перших та необхідних знань, як вони будуть використовувати протягом усього життя. Після школи учні стають студентами університетів, технікумів, коледжів та інших закладів освіти. Але і отриманням диплому про вищу освіту не закінчується відвідування уроків та занять.

В сучасному світі, де конкуренція за місце роботи велика, людям необхідні унікальні знання, які б допомогли їм виділитися серед інших кандидатів. Це, наприклад, знання іноземних мов, знання та вміння застосовувати практик ефективного управління персоналом, та багато інших навичок. Тому люди відвідують різні курси та тренінги, де набувають знання, що допоможуть їм у майбутньому.

Тому високий рівень ефективності начального процесу найважливіший фактор у здобутті нових знань. Ефективність процесу навчання може залежати від багатьох умов. І однією з таких умов є взаємодія та комунікація між викладачами та студентами.

1.1. Аналіз проблеми комунікації викладачів та студентів

Сучасні досліди показують, що викладачі можуть ігнорувати важливість комунікативних задач і не звертати уваги на необхідність педагогічного спілкування як умова успішного навчання [1].

Педагогічне спілкування – це процес комунікації викладача та студента під час заняття та за його межами, направлене на створення сприятливих умов для навчання, оптимізуючи навчальний процес та допомагати ефективному засвоєнню вивченого матеріалу [2].

Можливість задати питання викладачу та обговорити тему, що викладається, значно підвищує зацікавленість студентів у матеріалі та ефективність його засвоєння.

Тому була поставлена задача розробити систему, що націлена на поліпшення взаємодії під час лекцій, уроків, занять чи семінарів. Проблема, яку повинна вирішувати розроблювана система, це комунікація між викладачами та студентами під час занять.

1.2. Програмне рішення проблеми комунікації викладачів та студентів

Для доступу та взаємодії з системою буде розроблений програмний продукт, що представляє собою веб-додаток, що є універсальним та кросплатформенним рішенням. Доступ до веб-додатку можна отримати через браузер, який можна запустити на будь-якому пристрої, що має доступ до мережі Інтернет, та з будь якою операційною системою.

За допомогою програмного продукту студенти зможуть задати будь яке питання під час заняття або ж після нього, якщо йому щось не зрозуміло. Це допоможе не відволікати викладача від лекції, та дозволить йому відповісти у зручний для нього час. Також студенти зможуть продивитися усі питання, що були задані раніше. Інші студенти також зможуть долучитися до обговорення будь-якого питання та спробувати дати на нього відповідь.

Викладач буде мати доступ до усіх питань, щоб відповісти на них, або відмітити правильну відповідь, якщо хтось з студентів відповів до цього.

Користувачі будуть мати можливість продивитися список предметів та історію лекцій для кожного з них, щоб знайти питання та відповіді до цікавої для них теми.

Програмний продукт націлений на студентів та викладачів вищих начальних закладів, шкіл, курсів та інших закладів освіти.

1.3. Огляд існуючих програмних рішень для поліпшення проведення занять

Дослідження вже існуючих програмних систем є дуже важливим етапом в плануванні та формулюванні точного технічного завдання.

Було проведено огляд існуючих систем або застосунків для поліпшення проведення занять, але було виявлено, що даний момент аналогічних систем або додатків майже не існує. Було знайдено декілька додатків, що представляють собою електронний щоденник, але у них відсутня синхронізація з іншими користувачами для доступу до спільних занять, тобто призначених тільки для особистого використання.

Існує система «Електронний кампус», розроблена для організації навчального процесу, яка дозволяє оцінювати роботу студентів та мати доступ до матеріалів лекції. Але там відсутній доступ до лекції у режими онлайн з можливістю обговорювати тему та задавати питання. Ця система не вирішує проблему комунікації між студентами та викладачами.

Деякі курси мають власні додатки для комунікації між студентами та викладачами, але їх недоліком є те, що вони доступні лише для студентів конкретного закладу. Це робить недоступним їх використання студентам у інших закладів.

Також існує велика кількість застосунків для обміну повідомленнями, але вони не мають ніякої організації по лекціям та предметам, що робить їх абсолютно незручними для використання під час навчального процесу.

Тому можна зробити висновок, що розроблюваний програмний продукт надає унікальний функціонал, що дозволить студентам та викладачам взаємодіяти більш зручним способом.

2. ЗАСОБИ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ПРОДУКТУ СИСТЕМИ ПОЛІПШЕННЯ ЗАНЯТЬ

Аналізуючи поставлену задачу та методи її вирішення, було вирішено розробити клієнт-серверну систему, що складається з веб-додатку, який буде працювати у зв'язці з хмарною базою даних, що дозволить користувачам мати доступ з будь-якого пристрою, що має доступ до мережі Інтернет через браузер.

2.1. Клієнт-серверна архітектура

Веб-застосунки зазвичай представляють собою два комп'ютера, що спілкуються між собою – клієнт та сервер. Цей концепт зображено на рисунку 2.1.

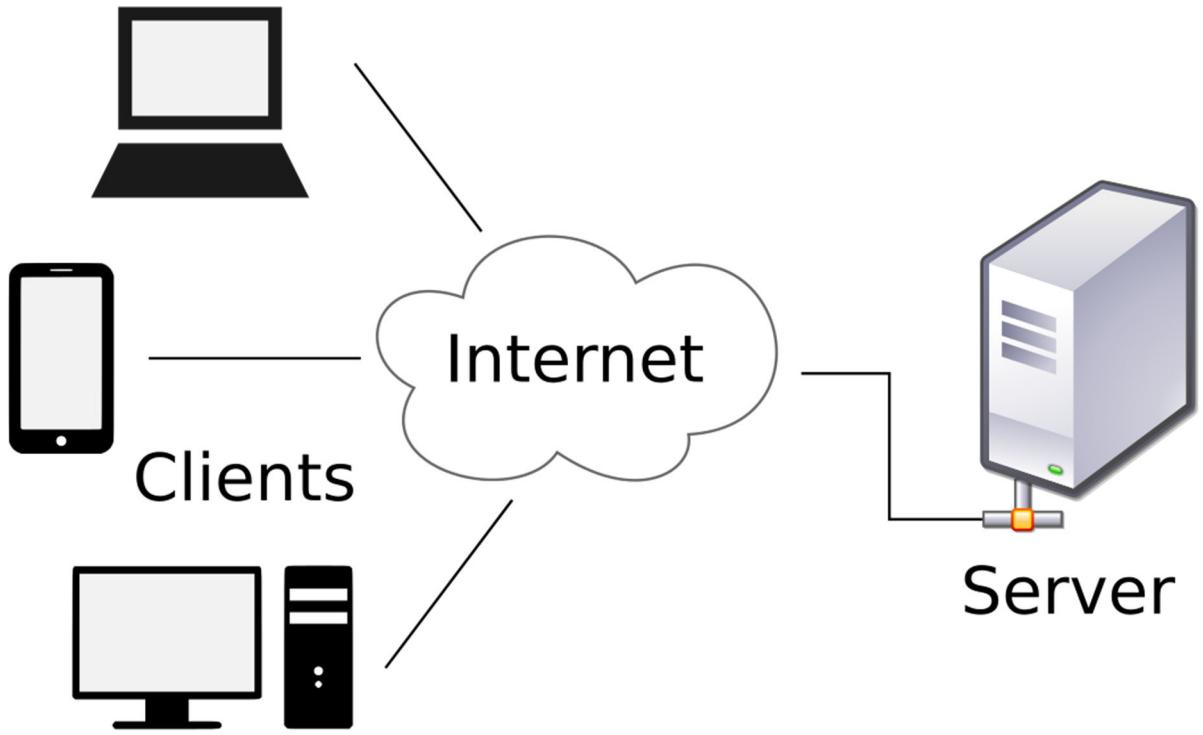

Рисунок 2.1 – Клієнт-серверна архітектура

Взаємодія між клієнтом та сервером відбувається за допомогою через HTTP (HyperText Transfer Protocol) [3]. Протокол HTTP представляє собою стандарт, що регулює порядок направлення запитів та відповідей – процесу, що відбувається між браузером, що запущено на комп'ютері кінцевого користувача та веб-сервером.

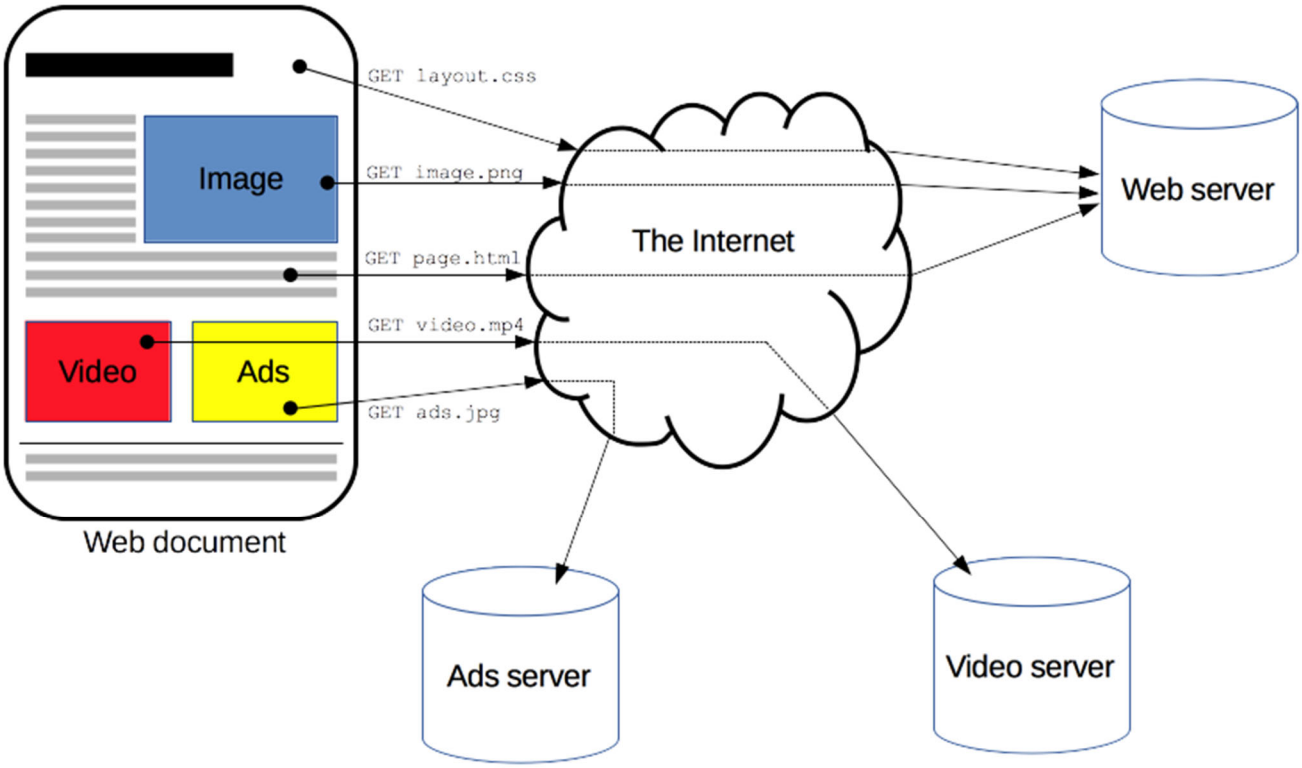

Рисунок 2.2 – Протокол HTTP

За допомогою URL, клієнт точно визначає адресу сервера, до якого потрібно зробити запит. За допомогою HTTP методу визначається, яку дію необхідно виконати.

Основні методи:

- GET – отримати дані з цього ресурсу;
- POST – створити новий ресурс, використовуючи дані, передані у тілі запиту;
- PUT – оновити існуючі дані, нові дані присутні у тілі запиту;
- DELETE – видалити вказаний ресурс.

Браузер – це програмне забезпечення, що дозволяє взаємодіяти з контентом веб-сторінок, отриманих через мережу Інтернет. Браузер використовує протокол HTTP для взаємодії з сервером та отримання від нього необхідної інформації.

Сервер – це комп'ютер, що може знаходитися у приміщені компанії або дата-центрі й приймає запити від клієнта та надсилає йому відповіді. Також сервер має доступ до даних, що використовуються застосунком [8].

Користувачі використовують свої браузери – клієнти – для взаємодії з додатком. Вони відправляють HTTP запити, які обробляються сервером, та отримують від нього відповідь.

2.2. Архітектура сервера та бази даних

Для обробки та зберігання даних було обрано сервіс хмарового сховища даних Firebase, який надає базу даних та сервер як службу. Цей сервіс надає розробникам API, яке дозволяє мати доступ до даних через клієнтський застосунок та синхронізувати їм між клієнтами та зберігати їх у хмарі.

Використання хмарового сховища дозволяє зекономити на серверній інфраструктурі та мати зручний доступ до даних.

Клієнтський додаток буде підключатися до хмарового сховище через HTTP протокол та за допомогою REST API взаємодіяти з ним.

2.3. Архітектура клієнтського додатку

На даний момент існує два підходи у розробці клієнтських додатків: односторінкові застосунки (SPA) та серверний додаток (SSA).

Серверні додатки оброблюються на боці сервера, тобто клієнт використовується тільки щоб відобразити сторінку, попередньо згенеровану на сервері [10]. Коли користувач взаємодіє з додатком, клієнт відправляє відповідний запит до сервера, який виконує задані операції та повертає нову згенеровану сторінку

у відповідь. Тому сторінка постійно перезавантажується кожного разу, щоб відобразити актуальні дані (рисунок 2.2).

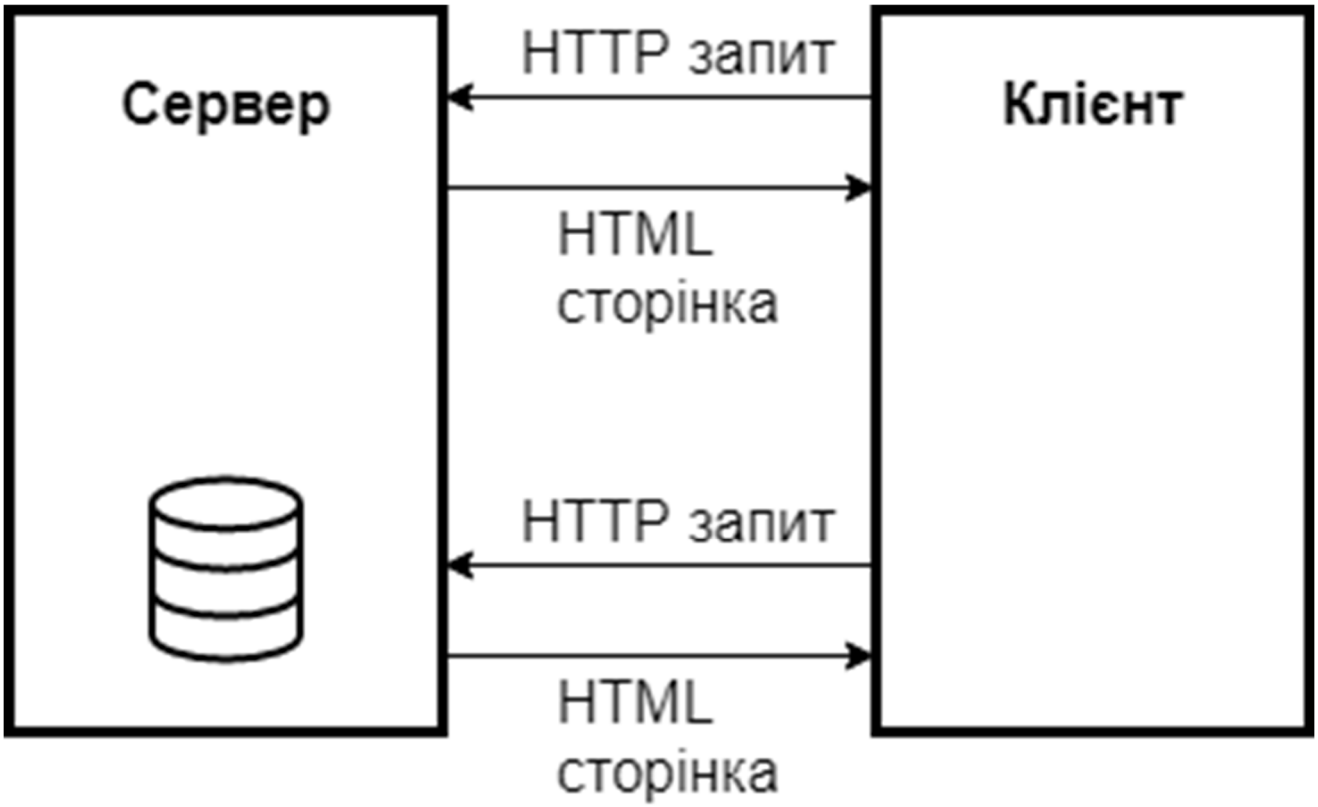

Рисунок 2.3 – Схема роботи серверного застосунку

Односторінковий застосунок (SPA) – це веб-застосунок чи веб-сайт, який вміщується на одній сторінці з метою забезпечити користувачу досвід близький до користування настільною програмою.

Технологія односторінкових веб-додатків більш новий феномен і зараз комп’ютерна індустрія рухається у напряму такої моделі. При такій архітектурі більша частина роботи все ще виконується на сервері, але частина коду виконується на стороні клієнта – у браузері користувача. Це допомагає уникнути постійного перезавантаження сторінки.

Коли користувач взаємодіє з додатком, він надсилає запит до сервера, який у свою чергу оброблює запит да відсилає лише необхідні данні у вигляді формату

JSON. Клієнт отримує ці дані та динамічно змінює відображення в залежності від них (рисунок 2.3).

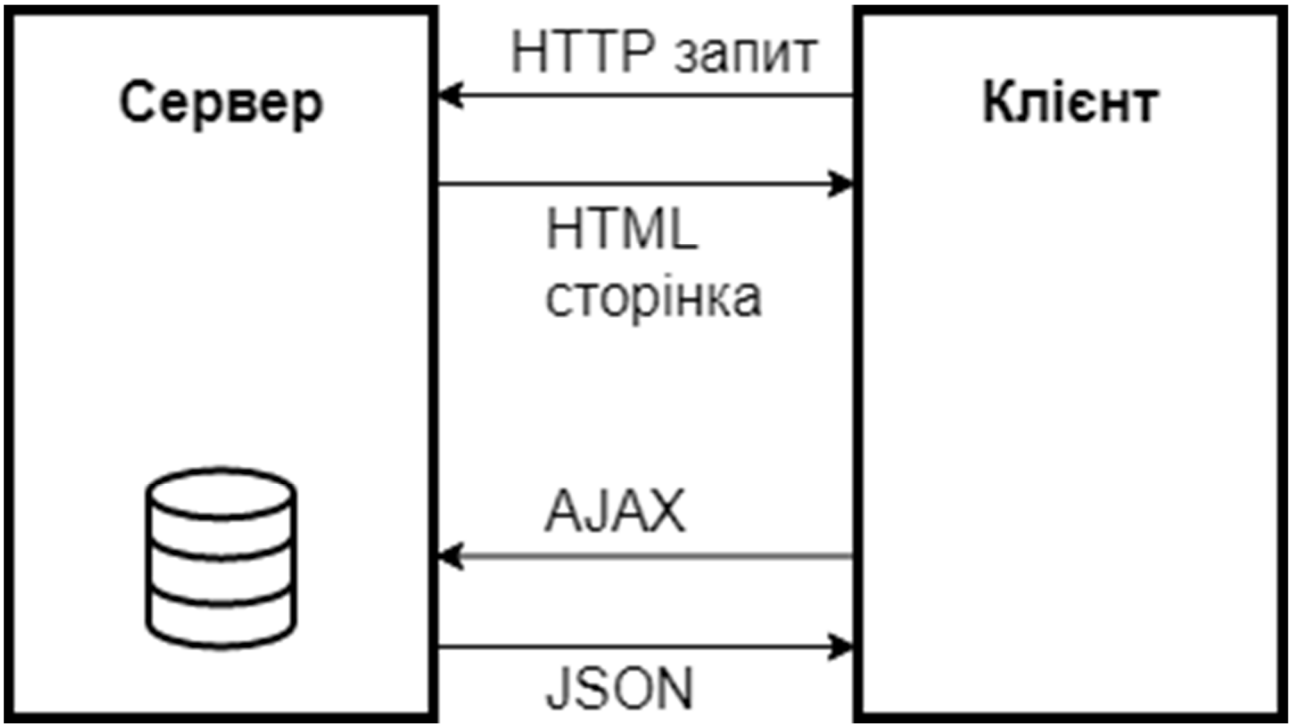

Рисунок 2.4 – Схема роботи односторінкового застосунку

Для розробки клієнтського веб-додатку було обрано SPA архітектуру.

Перевага такого підходу у розробці веб-додатків полягає у тому, що весь необхідний код завантажується при ініціалізації разом зі сторінкою один раз, або динамічно довантажується за потребою, зазвичай у відповідь на дії користувача. При подальшій взаємодії завантажуються лише дані з сервера та відображаються в залежності від логіки програми. Сторінка не оновлюється і не перенаправляє користувача до іншої сторінки у процесі роботи з нею. Це прискорює швидкість навігації та зменшує навантаження на сервер.

2.4. Технологія AJAX

Взаємодія між клієнтським кодом та сервером відбувається за допомогою технології AJAX. AJAX – це стандарт асинхронних запитів JavaScript [9].

Коли клієнту потрібно зробити запит до сервера, він використовує AJAX, щоб надіслати потрібні дані та чекає результатів, що має повернути сервер. Результат включає в себе дані, але не нову сторінку. Коли запит відісланий і клієнт чекає на відповідь, виконання коду не зупиняється, тому що він все ще повинен відобразити користувацький інтерфейс. В цьому полягає асинхронна частина AJAX.

Раніше AJAX використовував формат даних XML для запитів та відповідей, що обмінювалися між клієнтом та сервером.

Але зараз технологія AJAX використовує формат даних JSON замість XML, тому що цей формат більш компактний та зручний для сприйняття людиною.

2.5. Архітектурний шаблон MVC

Модель–представлення–контролер (MVC) – це архітектурний шаблон проектування для реалізації клієнтських інтерфейсів [5].

Шаблон поділяє програму на три частини, що зв’язані та взаємодіють між собою:

- Модель – відповідає за збереження, опис та обробку даних, що використовує програма.
- Представлення – відповідає за відображення даних користувачу, тобто описує користувацький інтерфейс.
- Контролер – зв’язує модель та представлення, отримує дані від моделі та повертає їх у шаблон, або викликає команди моделі у відповідь на взаємодію з представленням.

Схема шаблона відображена на рисунку 2.4.

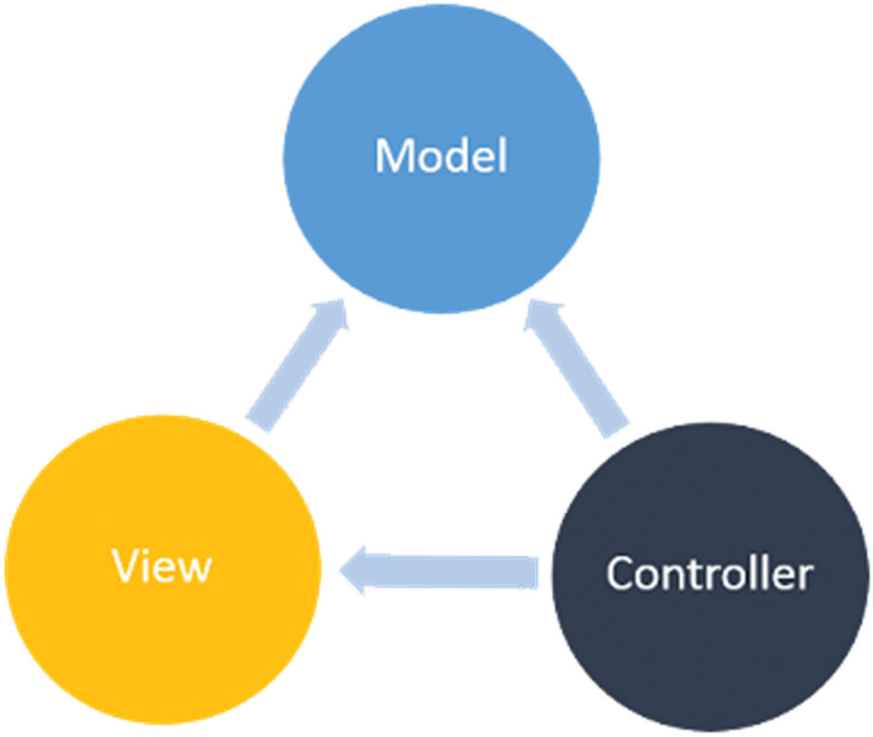

Рисунок 2.5 – Шаблон MVC

Шаблон MVC описує взаємодію між цими частинами трьома частинами програми.

Представлення відображає данні у моделі, але не може їх змінювати напряму у моделі. Представлення використовує контролер для виклику методів, що змінюють дані.

Контролер оновлює модель та оновлює представлення відповідно до нової інформації.

Модель описує дані, що використовуються у програмі. Модель може зберігати логіку, безпосередньо зв’язану з цими даними, але в той же момент модель не повинна зберігати ніякої логіки, пов’язаної з відображенням даних і взаємодією з елементами управління.

2.6. Опис інструментів розробки

Для розробки клієнтського застосунку використовується фреймворк Angular, який реалізує шаблон MVC.

Розробкою фреймворку Angular займається компанія Google, яка представила його у 2016 році. Angular надає таку функціональність, як двостороння прив’язка даних, що дозволяє змінювати данні в одному місці інтерфейсу при зміні моделі у іншому, шаблони, маршрутизація та інше.

Одною з особливістю Angular є те, що він використовує в якості мови програмування TypeScript.

TypeScript — мова програмування, представлена Microsoft восени 2012; позиціонується як засіб розробки веб-застосунків, що розширює можливості JavaScript. TypeScript є зворотньо сумісним з JavaScript. Фактично, після компіляції програму на TypeScript можна виконувати в будь-якому сучасному браузері або використовувати спільно з серверною платформою Node.js [7].

Переваги над JavaScript:

- можливість явного визначення типів (статична типізація).
- підтримка використання повноцінних класів.
- підтримка підключення модулів.
- інтерфейси, абстрактні класи
- закриті поля

2.7. Архітектура фреймворку Angular

Фреймворк Angular – це платформа для створення клієнтських додатків за допомогою HTML та TypeScript. Він реалізує базову да додаткову функціональність як набір бібліотек, які можна імпортувати у програмний продукт.

Використання фреймворку полегшує управління архітектурою коду, шаблонами проектування та зменшує час розробка програмного забезпечення.

Більшість сучасних програмних рішень розробляється з використанням фреймворків та бібліотек.

Фреймворки дозволяють структурувати програмний код та змушують писати його визначеним способом. Це дозволяє підтримувати весь код у єдиному стилі, щ полегшує роботу інших розробників з чужими частинами програмного коду.

Бібліотеки зазвичай пропонують набір функцій та API, які можуть бути використані розробником у коді.

Приклади фреймворків:

- Angular - фреймворк з відкритим вихідним кодом, призначений для розробки веб-додатків. Спрощує створення призначених для користувача компонентів, які можуть бути додані в документи HTML, а також реалізацію логіки докладання. Активно використовує прив'язку даних, містить модуль впровадження залежності, підтримує модульність і надає механізм для налаштування маршрутизації.
- Ember.js - це фреймворк з відкритим вихідним кодом, заснований на MVC; служить для розробки веб-додатків. Містить механізм маршрутизації і підтримує двосторонню прив'язку даних. У коді цього фреймворку використовується безліч угод, що підвищує продуктивність розробників ПЗ.
- Jasmine - фреймворк з відкритим вихідним кодом, призначений для тестування коду JavaScript. Не вимагає наявності об'єкта DOM. Містить набір функцій, які перевіряють, чи ведуть себе частини програми запланованим чином. Нерідко використовується разом з Karma - програмою для запуску тестів, яка дозволяє проводити перевірки в різних браузерах.

Прикладами бібліотек для розробки веб додатків можуть бути:

- jQuery - популярна бібліотека для JavaScript. Досить проста у використанні і не вимагає внесення суттєвих змін до стилю написання коду для вебпрограмм. Дозволяє знаходити об'єкти DOM і маніпулювати ними, а також обробляти події браузера і справлятися з несумісністю браузерів. Це розширювана бібліотека, розробники з усього світу створили для неї тисячі плагінів.

- Bootstrap - бібліотека компонентів для створення призначеного для користувача інтерфейсу з відкритим вихідним кодом, розроблена компанією Twitter. Вони будуються відповідно до принципів адаптивного веб-дизайну, що значно підвищує цінність бібліотеки, якщо веб-додаток повинен автоматично підлаштовувати свій макет в залежності від розміру екрану пристрою користувача.
- Material Design - бібліотека компонентів, розроблена у компанії Google, яка може стати альтернативою Bootstrap. Вона оптимізована для використання на різних пристроях і поставляється разом з набором різних елементів призначеного для користувача інтерфейсу.
- React - створена компанією Facebook бібліотека з відкритим вихідним кодом, призначена для збирання призначених для користувача інтерфейсів. Являє собою шар V в абревіатурі MVC. Створює власний віртуальний об'єкт DOM, мінімізуючи доступ до об'єкта DOM браузера, в результаті чого підвищується продуктивність. Що стосується відтворення вмісту, React вводить формат JSX - розширення синтаксису JavaScript, який виглядає як XML.
- Polymer - бібліотека, створена компанією Google для збірки призначених для користувача компонентів на основі стандарту Web Components. Поставляється разом з набором цікавих настроюються елементів призначеного для користувача інтерфейсу, які можна включити в розмітку HTML у вигляді тегів. Крім того, містить компоненти додатків, призначених для роботи в режимі офлайн, а також елементи, які використовують різноманітні API від Google (наприклад, календар, карти та ін.).
- RxJS - набір бібліотек, необхідних для створення асинхронних програм і програм, заснованих на подіях, з використанням спостережуваних колекцій. Дозволяє програмам працювати з асинхронними потоками даних на зразок серверного потоку котирувань акцій або подій, пов'язаних з рухом миші. За допомогою RxJS потоки даних представляються у вигляді спостережуваних послідовностей. Цю бібліотеку можна застосовувати як з іншими фреймворками JavaScript, так і без них.

Базовими структурними блоками Angular є модулі, які надають контекст компіляції для компонентів. Модулі зберігають залежний код як функціональний

набір, а програма - це набір модулів. Програма завжди має мінімум один модуль – кореневий, який виконує завантаження додатку, та, зазвичай, багато функціональних модулів [6].

Компоненти визначають відображення, яке являє собою набір елементів екрану, які Angular може відображати та змінювати в залежності від логіки програми та даними.

Компоненти використовують сервіси, які надають функціональність, яка напряму не відноситься для відображення. Сервіси можуть підключатися до компоненту як залежності, що робить код модульним та зручним для багаторазового використання. І компоненти, і сервіси – це просто класи, які мають декоратори з необхідними метаданими, які вказують фреймворку, як їх використовувати (рисунок 2.5).

Метадані класів компонентів пов’язані з їх шаблоном, що визначають представлення. Шаблон поєднує звичайний HTML разом з директивами та шаблоном прив’язки даних, що дозволяє Angular змінювати HTML перед відображенням шаблону.

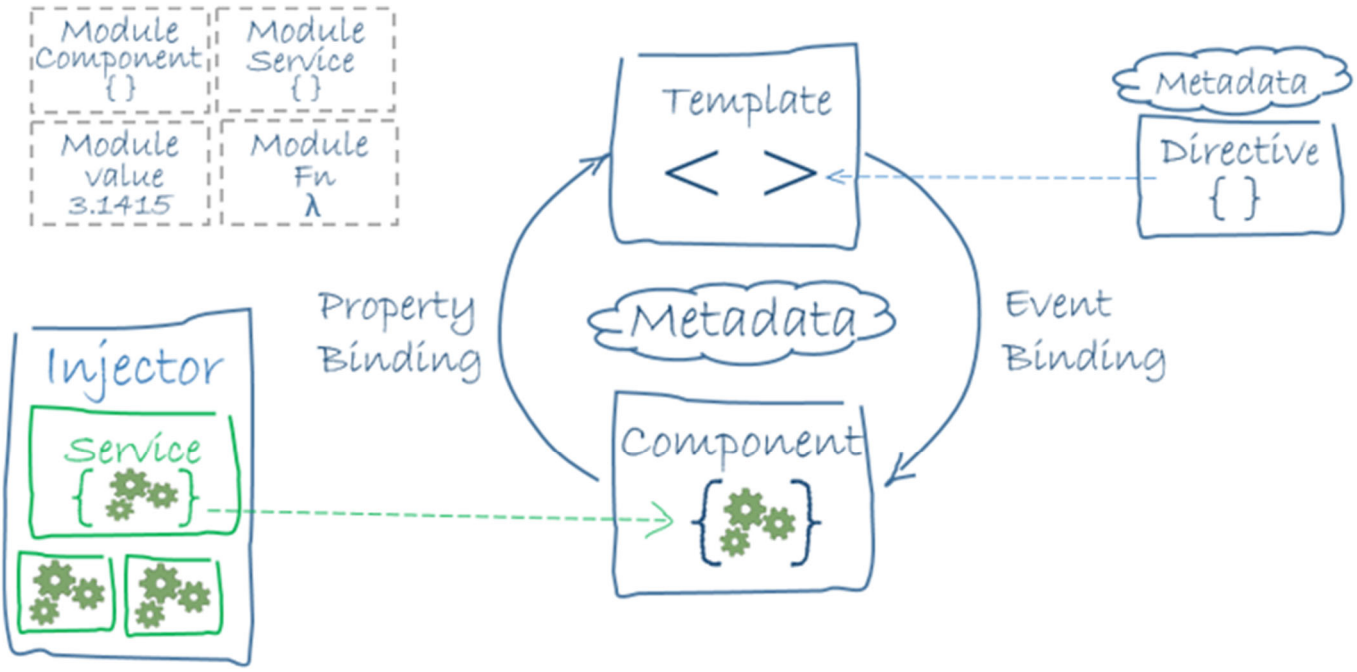

Рисунок 2.6 – Архітектура Angular програми

Метадані класів сервісів надають інформацію, яку потребує Angular, щоб зробити їх доступними для усіх компонентів завдяки технології впровадження залежності.

Компоненти зазвичай визначають багато шаблонів відображення, що зв'язані ієрархічно. Тому Angular надає сервіс маршрутизації, що допомогає визначити шлях навігації до того чи іншого відображення.

2.7.1. Модулі

Програма Angular складається з окремих модулів. Зазвичай, програми складаються з декількох модулів. Модуль – це контейнер для пов'язаних блоків коду, які виконують схожі задачі або близько пов'язані обов'язками, що вони виконують у програмі.

Вони можуть зберігати компоненти, надавати сервіси та інші файли коду, чия область визначена цим модулем. Вони можуть імпортувати функціональність з інших модулів та експортувати деяку свою функціональність, що дозволяє використовувати її повторно.

Модулі надають контекст компіляції для своїх компонентів. Кореневий модуль має кореневий компонент, який створюється при завантаженні програми, але модулі можуть мати велику кількість компонентів, які підключаються у шаблоні представлення або створюються в залежності від шляху маршрутизації.

Модулі визначаються за допомогою класів з анотацією @NgModule(). Анотація @NgModule() – це функція, що приймає матадату у вигляді об'єкту, що описує цей модуль (рисунок 2.6). Основін властивості модуля:

- declarations – описує компоненти, що належать до цього модулю;
- exports – компоненти, які можуть бути використані іншими модулями;
- imports – список модулів, компоненти яких можуть бути використані у межах цього модулю;
- providers – сервіси, що модуль надає до глобальної колекції сервісів;

- bootstrap – головний компонент програми, який відповідає за відображення головного вікна.

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
@NgModule({
  imports:      [ BrowserModule ],
  providers:    [ Logger ],
  declarations: [ AppComponent ],
  exports:      [ AppComponent ],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

Рисунок 2.7 – Приклад модуля

Кожна програма Angular має кореневий модуль – AppModule. Малі програми можуть мати тільки один модуль – кореневий, але більшість має багато і функціональних модулів.

2.7.2. Компоненти

Важливими елементами програми є компоненти. Компонент керує відображенням представлення на екрані. Кожен компонент – це TypeScript клас, що має відповідний декоратор, який вказує компілятору, що його слід використовувати як компонент.

Компоненти – це структурні блоки програми, які відповідають за логіку представлення. Крім коду TypeScript, компонент має свій шаблон HTML та стилі. У шаблоні зберігається розмітка, яка вказує браузеру, як відображати дані компонента.

Компоненти використовують дані з моделі, щоб передати їх у шаблон – представлення.

Компоненти використовують прив’язку даних між представленням та моделлю представлення, що дозволяє оновити дані, які відображаються, при оновленні даних у моделі.

Користувач може взаємодіяти з представленням, через візуальні елементи інтерфейсу: кнопки, поля форми тощо. Компонент реагує на дії користувача через механізм відслідковування подій та оновлює дані моделі.

Анотація @Component ідентифікує клас, для якого вона була об’явлена як компонент та вказує йому необхідні метадані.

Метадані для компонента вказують Angular де знаходяться основні блоки, які необхідні для створення та відображення компонента та його представлення. Компонент пов’язаний з шаблоном, який може бути створений у самому компоненті у вигдяді строки, або за посиланням на відповідний HTML файл, який зберігає відображення, що відповідає даному компоненту.

Крім шаблону, метадані компонента конфігурують, яким чином компонент може бути викликаний у інших HTML файлах або які сервіси він має використовувати.

Основні опції компонента:

- selector – CSS селектор, який вказує Angular створити та вставити цей компонент в будь-якому місці, де він був знайдений;
- templateUrl – відносний шлях шаблону, що відповідає цьому компоненту та відповідає за його відображення;
- providers – список сервісів, що необхідні компоненту.

Компонент визначається разом зі своїм шаблоном. Шаблон – це форма HTML, яка вказує Angular як відображати компонент.

Представлення зазвичай організовані ієрархічно, дозволяючи змінювати або відображати чи приховувати ті чи інші секції інтерфейсу (рисунок 2.7).

Шаблон виглядає як звичайний HTML, крім деяких специфічних елементів синтаксису Angular, які змінюють представлення в залежності від логіки програми та стану даних.

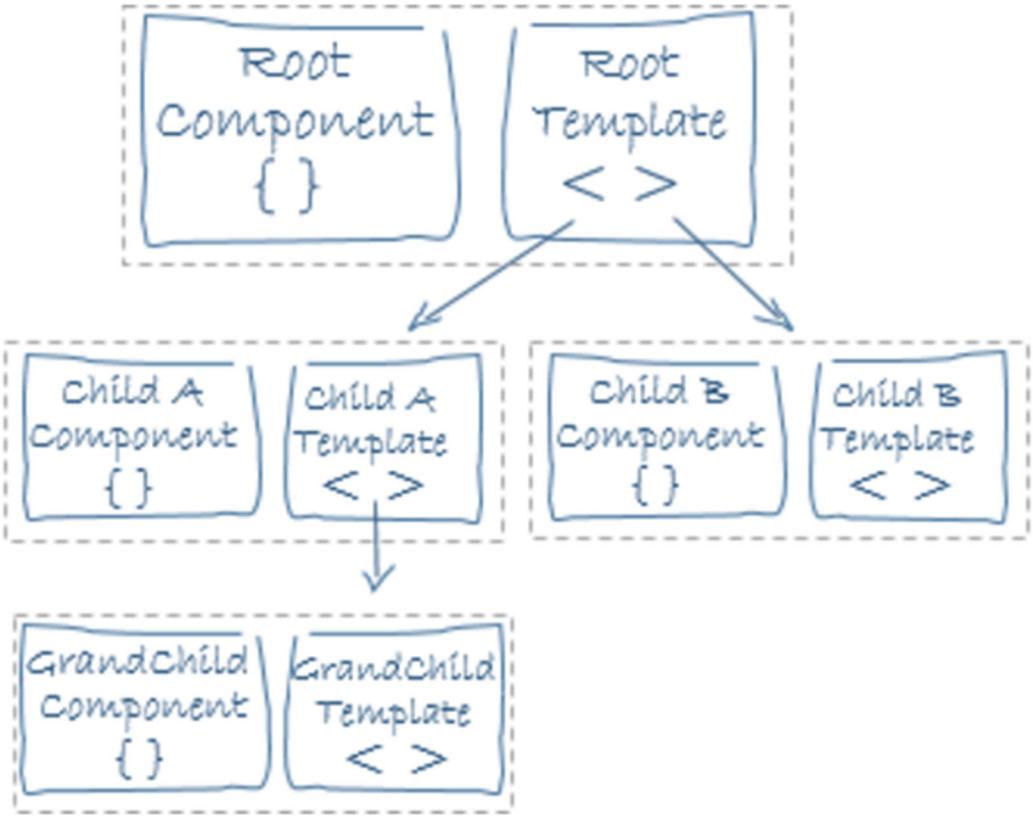

Рисунок 2.8 – Ієрархія компонентів

Шаблони Angular підтримують двосторонню прив’язку даних, механізм для координації частин шаблону та частин компоненту. Рисунок 2.8 відображає схему роботи двосторонньої прив’язки даних.

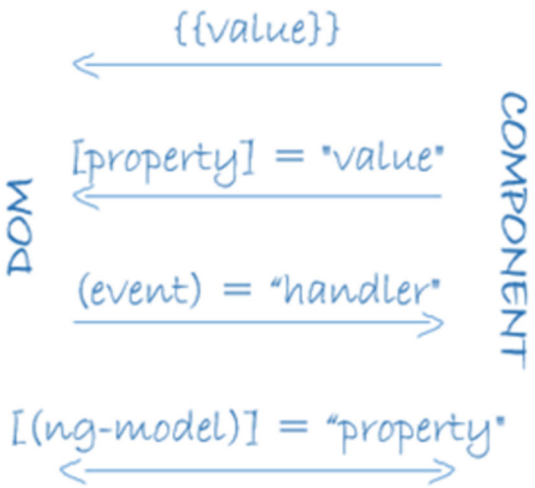

Рисунок 2.9 – Двостороння прив’язка даних

При змані стану компоненту Angular оновлює DOM відповідно з новими даними. Якщо ж відбулася подія, наприклад, користувач ввів дані у поле вводу, дані, прив’язані до нього автоматично оновлюються.

2.7.3. Сервіси

Сервіси надають широкий спектр класів, які виконують деякі специфічні задачі, наприклад, отримання даних з серверу, обробка даних.

На відміну від компонентів, сервіси не працюють з представленням. Іони виконують визначені задачі:

- Надання даних програмі, зберігання даних у пам’яті або взаємодія з сервером.
- Надання каналу зв’язку між компонентами.
- Інкапсуляція логіки задач, які не пов’язані з відображенням даних,.

Компоненти підключаються у модулях і компонентах, які використовують їх для отримання та оновлення даних. За допомогою механізму впровадження залежності, компоненти можуть використовувати лише один екземпляр сервісу, що гарантує, що компоненти будуть мати однакові дані.

Сервіси використовують механізм впровадження залежності для зберігання одного екземпляру у межах модуля.

Впровадження залежності широко використовується фреймворком Angular для використання сервісів у будь якому компоненті, де вони потрібні.

Для визначення класу як сервіс, потрібно використовувати @Injectable() анотація, щоб дозволити Angular впровадити його як залежність.

2.8. Веб-компоненти

Веб-компоненти представляють концепцію, яка охоплює чотири технології, розроблені для створення елементів з можливістю повторного їх використання, що веде до покращення модульності та полегшення обслуговування веб-програми.

Ці технології включають:

- Шаблони – фрагменти розмітки HTML, що визначають структуру відображення.
- Користувацькі елементи – шаблони, що містять не тільки традиційні HTML елементи, а й користувацькі, які розширяють їх функціонал.
- Тіньова модель DOM – інкапсулює правила CSS та модель поведінки елементів.
- Імпорт HTML – дозволяє розміщувати інші документи в HTML.

2.9. Бібліотека компонентів Angular Material

Для розробки клієнтського інтерфейсу було обрано використовувати бібліотеку компонентів Angular Material. Бібліотеки компонентів надають доступ до великої кількості типових елементів, які можна використати при розробці інтерфейсу клієнта веб-застосунку.

Бібліотека Angular Material реалізує принцип матеріального дизайну у своїх компонентах. На рисунку 2.10 зображено приклад матеріального дизайну.

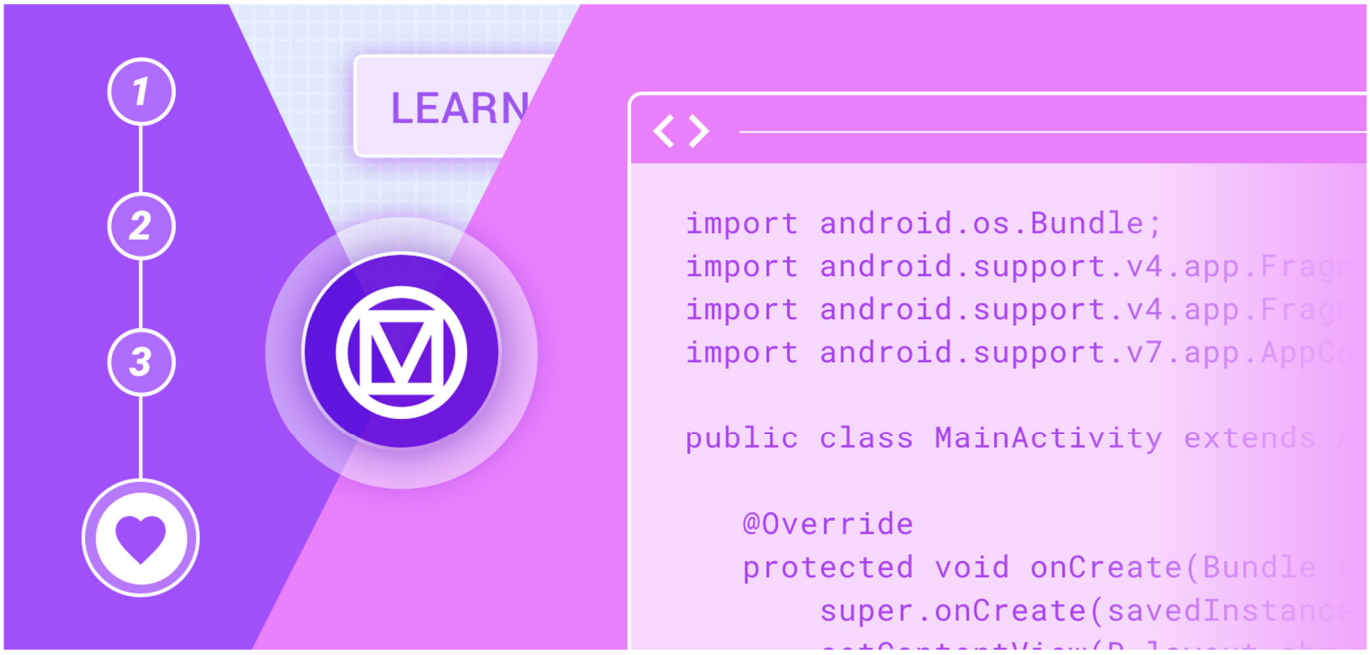

Рисунок 2.10 – Приклад елементів у матеріальному дизайні

Матеріальний дизайн – це сукупність підходів у розробці клієнтських інтерфейсів веб-додатків, мобільних застосунків та іншого програмного забезпечення, розроблені у компанії Google та представлені у 2014 році на конференції Google I/O.

Бібліотека Angular Material включає в себе велику кількість елементів користувацького інтерфейсу, основні з яких є:

- Компоненти кнопок;
- Елементи форм вводу даних;
- Компонент вибору дати;
- Компонент навігаційного меню;
- Таблиці відображення даних.

3. ОПИС ІНСТРУМЕНТІВ РОЗРОБКИ ДЛЯ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СИСТЕМИ ПОЛІПШЕННЯ ПРОВЕДЕННЯ ЗАНЯТЬ

Програма буде складатися з двох основних модулів – екрану авторизації та кабінету користувача, який доступний лише для авторизованих користувачів.

Модуль авторизації буде включати два екрани:

1. Форма реєстрації. На цьому екрані користувачі зможуть створити для себе обліковий запис для доступу до системи, якщо вони ще не є користувачами даного програмного продукту. Для реєстрації користувачі повинні будуть ввести свою електронну адресу, ім’я, телефон та придумати собі пароль. Всі поля є обов’язковими і будуть перевірятися, що б не допустити реєстрацію з неправильними даними.

2. Форма авторизації. На цьому екрані користувачі зможуть увійти до особистого кабінету за допомогою електронної адреси та пароля, які вони вказали при реєстрації.

Кабінет користувача буде включати два екрани:

1. Поточна лекція. На цьому екрані користувачі зможуть подивитися питання, що були задані під час поточної лекції та задати своє питання. Для кожного питання буде можливість продивитися існуючі відповіді та відповісти на нього. Для створення питання має бути відображена форма з полем вводу тексту нового питання.

2. Список предметів. На цьому екрані студенти будуть мати доступ до списку предметів, які для них доступні. Для кожного предмету будуть доступна історія лекцій, де користувачі зможуть побачити усі питання, які були задані під час цієї лекції та відповіді до них.

Детальна структура програми зображена на рисунку 3.1.

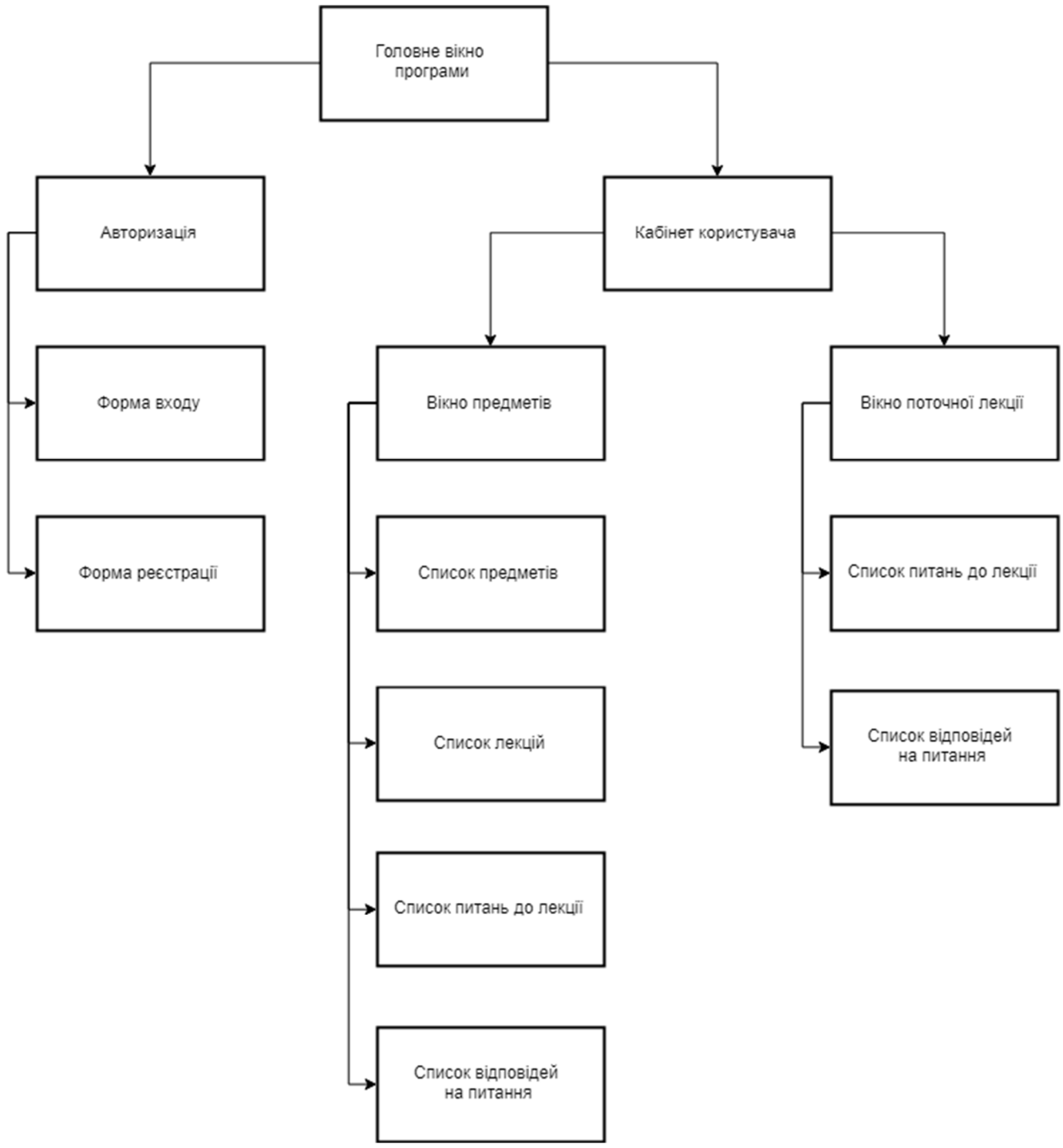

Рисунок 3.1 – Структура програми

Доступ до кабінету має бути тільки для авторизованих користувачів. При переході на кабінет неавторизованому пристрої, користувач має бути перенаправлений до сторінки авторизації.

3.1. Опис функціональності системи

На рисунку 3.2 зображено діаграму прецедентів, яка ілюструє можливі сценарії взаємодії користувачів з системою. У системі є два актори: Викладач та студент, які мають майже однакові можливості, за винятком того, що викладач має змогу відмічати правильні відповіді

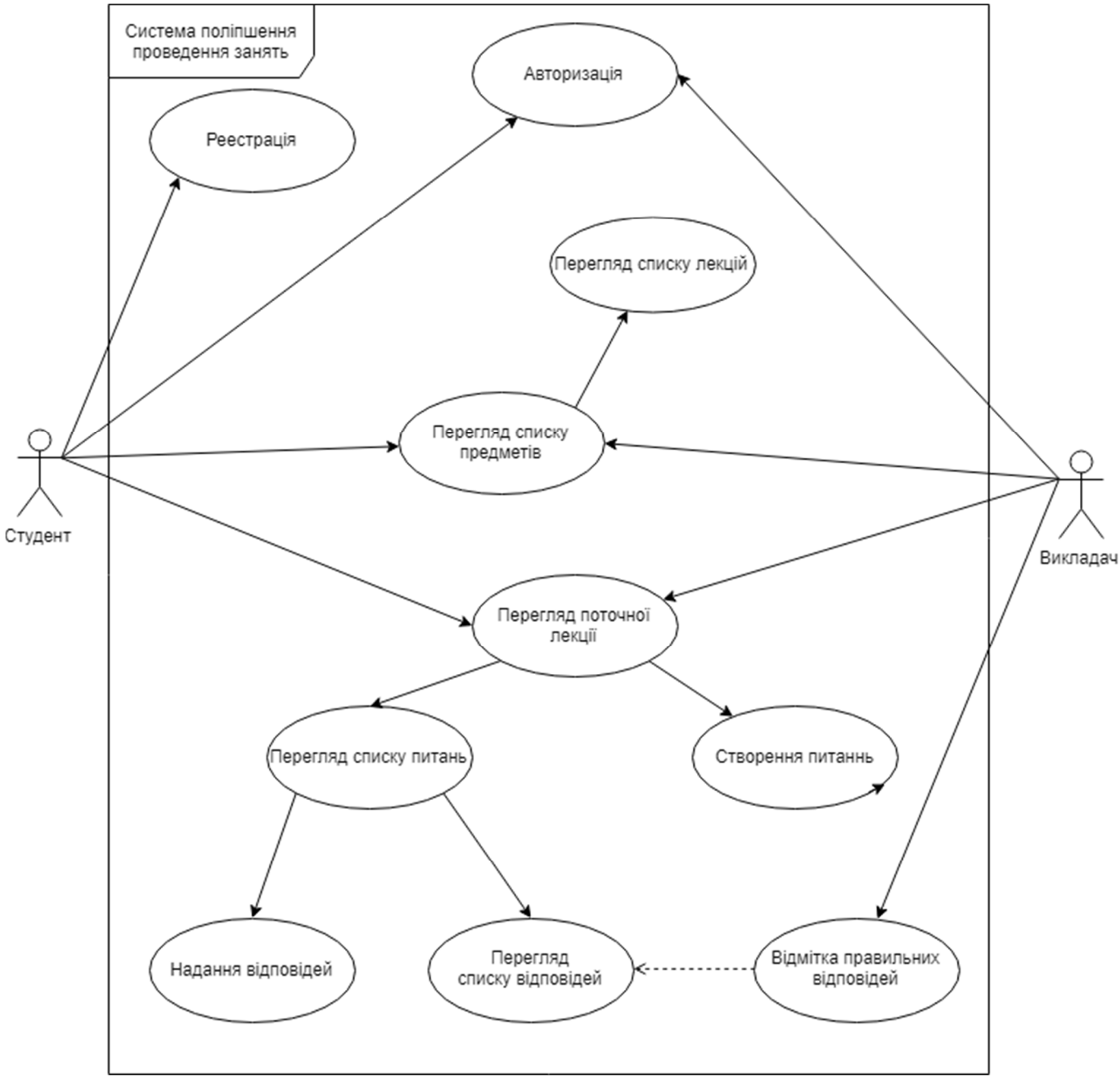

Рисунок 3.2 – Діаграма прецедентів

Як видно з рисунку, студент має такі функції:

- Створення обліковий запис у системі
- Вхід у систему, використовуючи дані, що були вказані при реєстрації
- Перегляд поточної лекції
- Перегляд заданих питань до поточної лекції
- Перегляд відповідей до обраних питань
- Перегляд списку предметів
- Перегляд списку лекцій до обраного предмету
- Перегляд питань до минулих лекцій

У викладача є можливість:

- Увійти у систему, використовуючи дані, що були вказані при реєстрації
- Переглянути поточну лекцію
- Переглянути задані питання до поточної лекції
- Переглянути відповіді до обраних питань
- Переглянути список предметів
- Переглянути список лекцій до обраного предмету
- Переглянути питання до минулих лекцій

3.2. Концептуальна модель бази даних

Концептуальна модель бази даних відображає предметну область, для якої створюється база даних. Вона допомагає зрозуміти, які сутності будуть використанні при розробці відповідного програмного продукту.

База даних системи поліпшення занять складається з п’яти сутностей: Користувач, Предмет, Лекція, Питання, Відповідь. На рисунку 3.3 зображена концептуальна модель їх зв’язку.

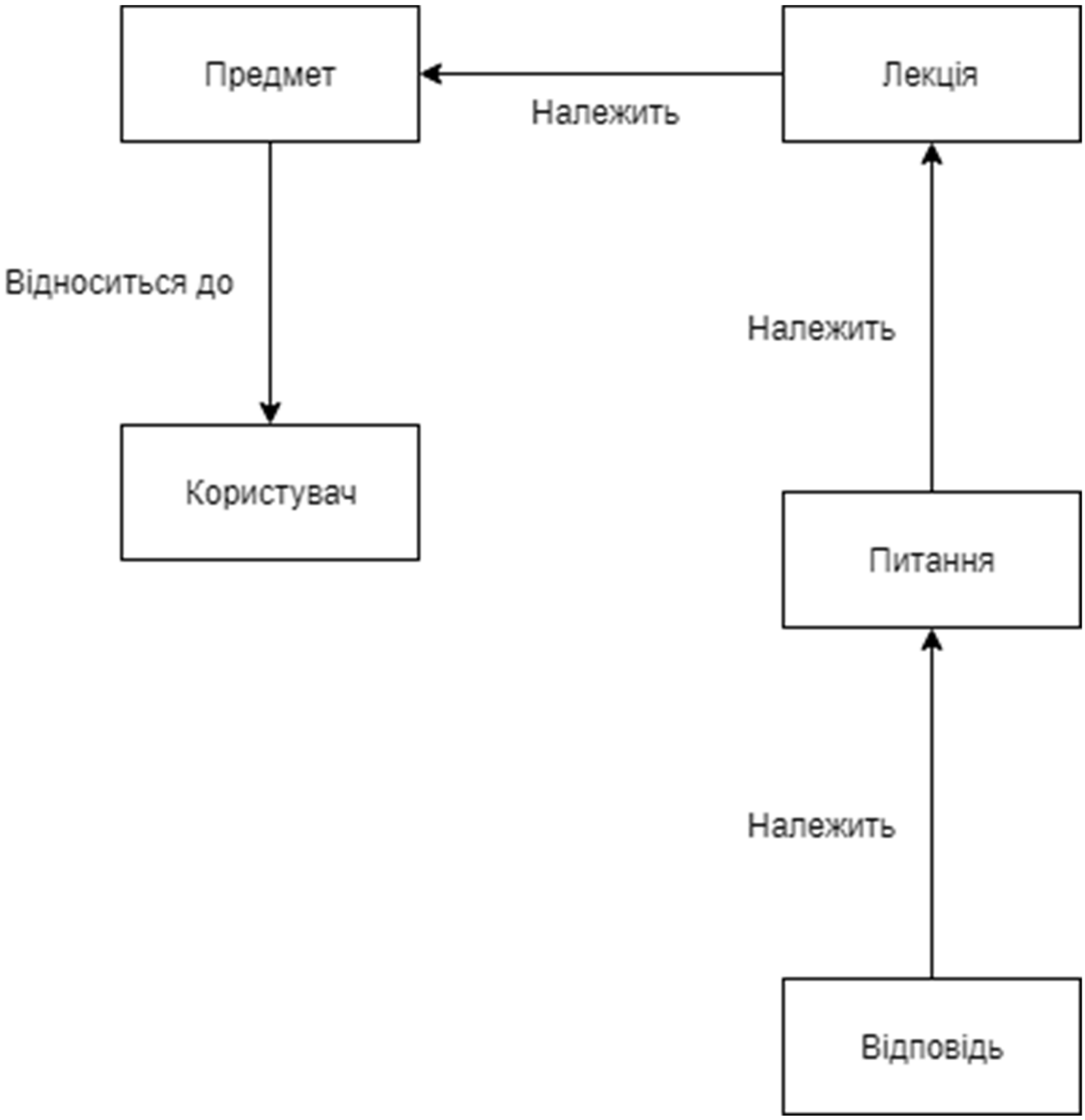

Рисунок 3.3 – Концептуальна модель бази даних

Як видно з схеми, кожна відповідь належить до якогось питання, яке в свою чергу належить до лекції. Кожна лекція належить предмету. Предмет відноситься о користувача, але кожен предмет може відноситися до декількох користувачі, а до кожного користувача може відноситися до декількох предметів.

3.3. Діаграма класів

Кожна сутність у базі даних має свій клас, який описує набір полів даних, що її належить. З рисунку 3.4 видно, які класи використовуються у системі, які типи яких даних описує кожен клас та як вони пов’язані між собою

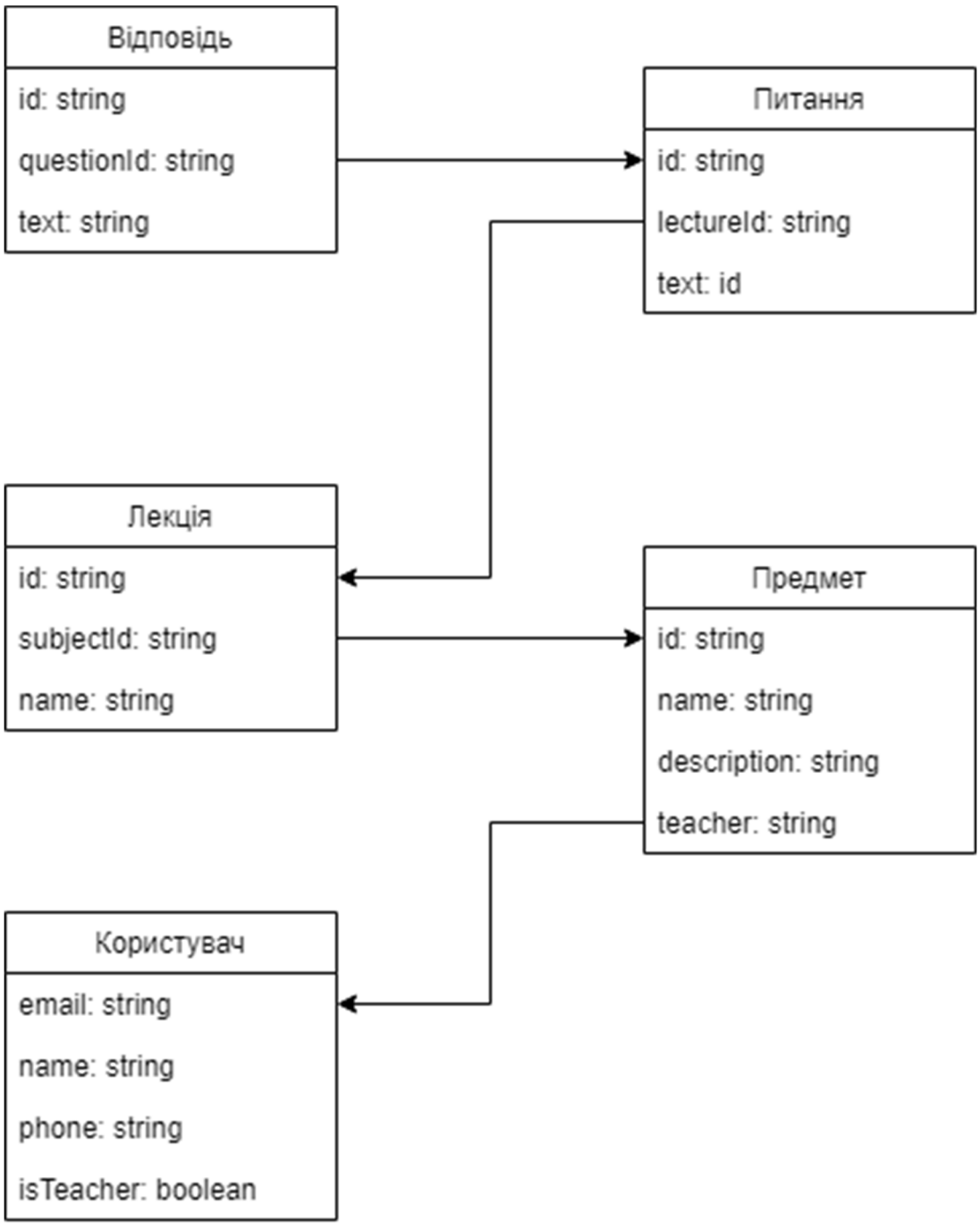

Рисунок 3.4 – Діаграма класів.

Як видно з рисунку, основою програмного продукту системи поліпшення занять є п’ять класів, що відповідають сутностям предметної області, для якої система розроблюється і якими система оперує.

Головною сутністю є Користувач, яка зберігає таку інформацію:

- email – електронна пошта користувача, яка водночас виступає унікальним ідентифікатором,
- name – ім’я та прізвище користувача,
- phone – номер мобильного телефона користувача,
- isTeacher – флаг, що вказує, чи є даний користувач викладачем.

Сутність Предмет зберігає інформацію про навчальний предмет:

- id –унікальний ідентифікатор предмета,
- name – назва предмета,
- description – додатковий опис предмета,
- teacher – поштова адреса користувача, що є викладачем даного предмета.

Сутність Лекція зберігає інформацію про лекцію:

- id –унікальний ідентифікатор лекції,
- name – назва лекції,
- subjectId – ідентифікатор предмета, до якого відноситься дана лекція.

Сутність Питання зберігає інформацію про питання до лекції, задане користувачем системи:

- id –унікальний ідентифікатор питання,
- text – текст питання,
- lectureId – ідентифікатор лекцыъ, до якої відноситься дане питання.

Сутність Відповідь зберігає інформацію про відповідь, дану на питання, що було задане до лекції:

- id –унікальний ідентифікатор відповіді,
- text – текст відповіді,
- questionId – ідентифікатор питання, до якого відноситься дана відповідь.

Класи системи описують предметну область да дані, які система зберігає та необхідні для її роботи.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ КЛІЄНТСЬКОГО ЗАСТОСУНКУ

Розробка програмного коду буде вестися у IDE (інтегроване середовище розробки) WebStorm (рисунок 4.1). Інтегроване середовище розробки полегшує розробку, надаючи розробнику такі інструменти, як автодоповнення коду, навігацію по коду, аналіз коду на синтаксичні помилки, інтеграцію з системою контролю версій.

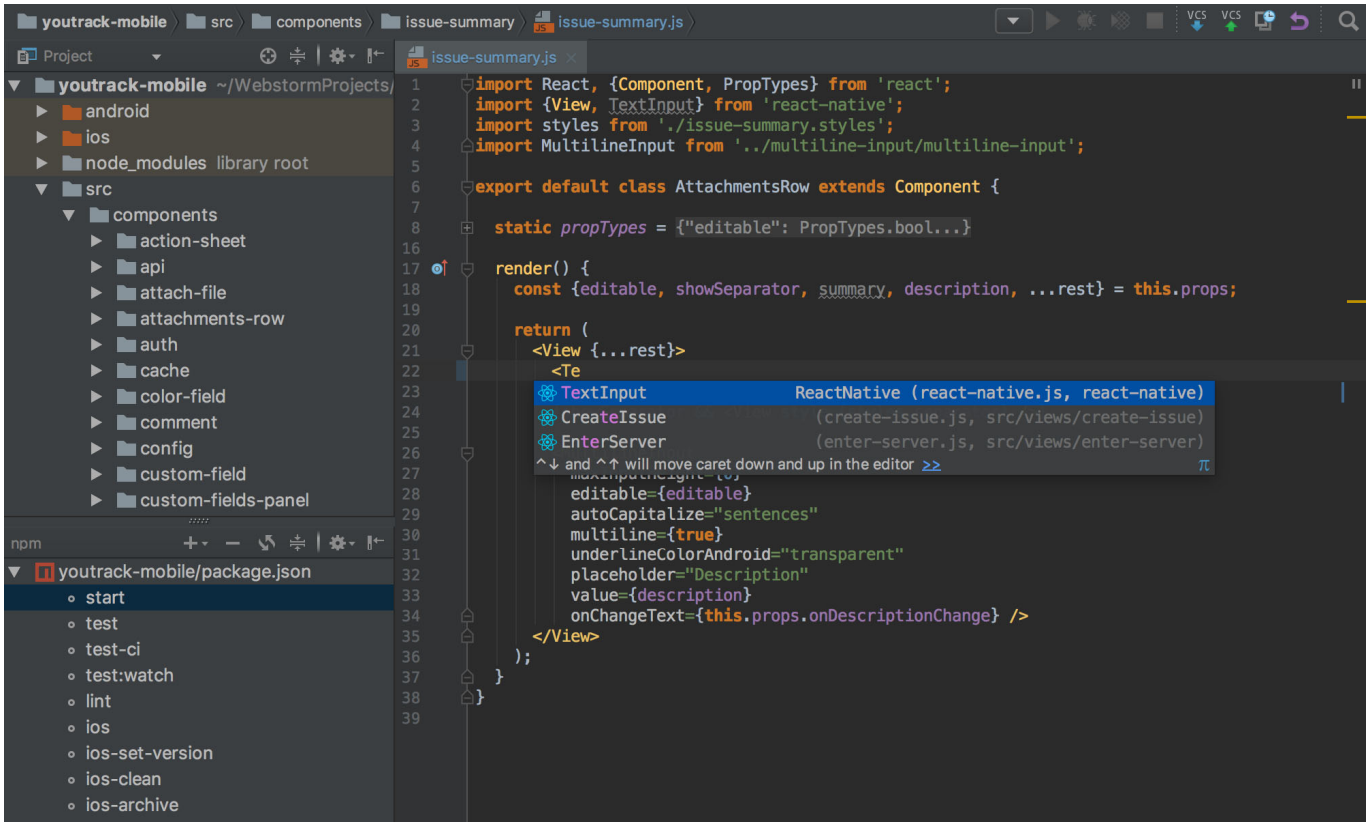

Рисунок 4.1 – Вікно IDE WebStorm

Середовище WebStorm має наперед встановленими плагінами для розробки веб-застосунків, які підтримують мови JavaScript, TypeScript, HTML, CSS та фреймворки для створення інтерфейсів.

4.1. Налаштування робочого оточення

4.1.1. Встановлення середовища Node.js

Розробка клієнту Angular потребує встановленого середовища Node.js. Node.js – це середовище часу виконання.

Фреймворк Node.js (рисунок 4.2) використовується для розробки програм на мові JavaScript, які функціонують поза браузера. Ви можете створити серверний шар веб-застосунку на JavaScript або TypeScript. Компанія Google розробила для браузера Chrome високопродуктивний рушій JavaScript V8. Його можна задіяти для запуску коду, написаного за допомогою API Node.js. Фреймворк Node.js включає в себе API для роботи з файловою системою, доступу до бази даних, прослуховування запитів HTTP.

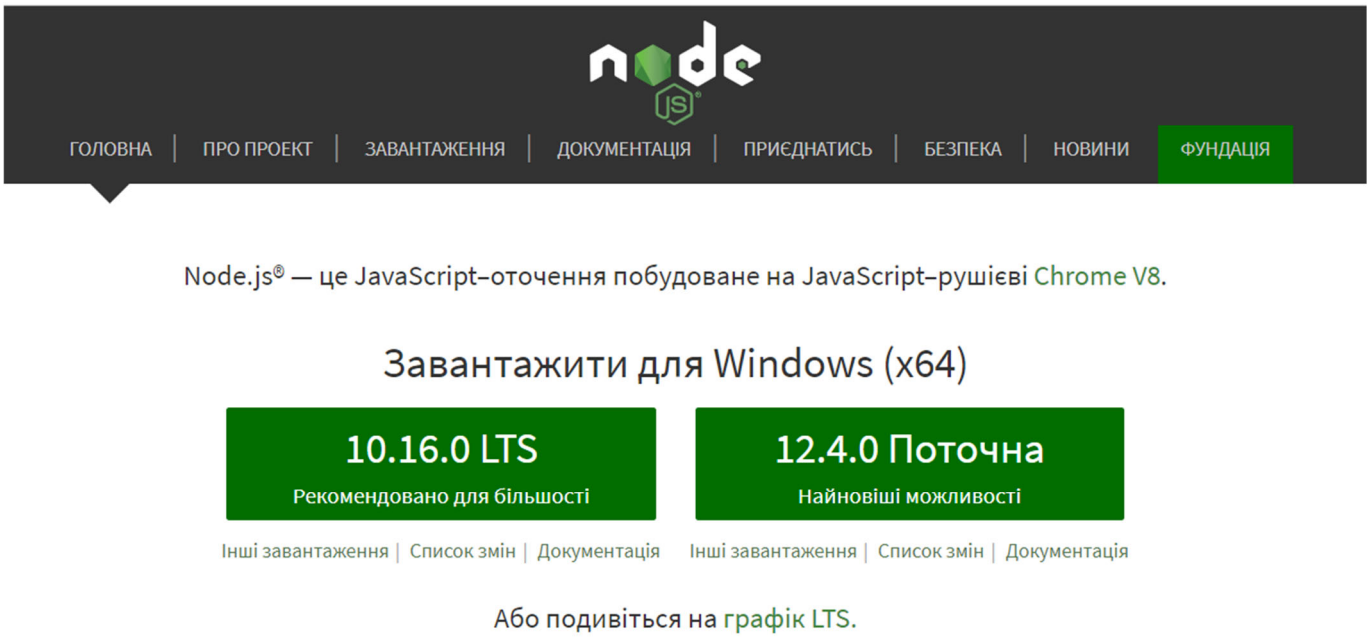

Рисунок 4.2 – Головна сторінка проекту Node.js

Після встановлення Node.js буде доступний пакетний менеджер npm, який допомагає встановити усі необхідні пакети. За допомогою npm можна встановити пакети, які знаходяться у віддаленому реєстрі пакетів як глобальний інструмент

JavaScript, так і локальні залежності програми чи інших пакетів, які використовуються.

4.1.2. Встановлення інструменту Angular CLI

Для створення нового проекту Angular можна використати інструмент Angular CLI (рисунок 4.3), що надає інтерфейс командного рядка для керування проектом, написаним на Angular.

Рисунок 4.3 – Встановлення та створення нового проекту

Angular CLI включає в себе набір команд для управління проектом, основні команди це:

- ng new – створити новий проект Angular;
- ng serve – запустити проект на локальній машині;
- ng build – компілює застосунок Angular;

- ng update – оновлює локальні пакети до останньої версії;
- ng version – виводить інформацію про поточну версію Angular CLI;
- ng g component – створює новий компонент;
- ng g module – створює новий модуль у програмі;
- ng g service – створює новий сервіс.

4.1.3. Створення нового проекту

Виклик команди ng new у терміналі створить новий проект у поточній директорії. Для створення проекту потрібно вказати його назву, обрати формат каскадних таблиць стилю та обрати опцію маршрутизації Angular веб-застосунку. Після цього у поточній директорії буде створена папка з назвою, яка була вказана раніше. Структура нового проекту зображена на рисунку 4.4

```
PS E:\> ng new angular-project
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? SCSS   [ http://sass-lang.com/documentation/
file.SASS_REFERENCE.html#syntax ]
CREATE angular-project/angular.json (3976 bytes)
CREATE angular-project/package.json (1314 bytes)
CREATE angular-project/README.md (1031 bytes)
CREATE angular-project/tsconfig.json (435 bytes)
CREATE angular-project/tslint.json (1621 bytes)
CREATE angular-project/.editorconfig (246 bytes)
CREATE angular-project/.gitignore (629 bytes)
CREATE angular-project/src/favicon.ico (5430 bytes)
CREATE angular-project/src/index.html (301 bytes)
CREATE angular-project/src/main.ts (372 bytes)
CREATE angular-project/src/polyfills.ts (2841 bytes)
CREATE angular-project/src/styles.scss (80 bytes)
CREATE angular-project/src/test.ts (642 bytes)
CREATE angular-project/src/browserslist (388 bytes)
CREATE angular-project/src/karma.conf.js (1028 bytes)
CREATE angular-project/src/tsconfig.app.json (166 bytes)
CREATE angular-project/src/tsconfig.spec.json (256 bytes)
CREATE angular-project/src/tslint.json (244 bytes)
CREATE angular-project/src/assets/.gitkeep (0 bytes)
CREATE angular-project/src/environments/environment.prod.ts (51 bytes)
CREATE angular-project/src/environments/environment.ts (662 bytes)
CREATE angular-project/src/app/app-routing.module.ts (245 bytes)
CREATE angular-project/src/app/app.module.ts (393 bytes)
CREATE angular-project/src/app/app.component.html (1152 bytes)
CREATE angular-project/src/app/app.component.spec.ts (1122 bytes)
CREATE angular-project/src/app/app.component.ts (220 bytes)
CREATE angular-project/src/app/app.component.scss (0 bytes)
CREATE angular-project/e2e/protractor.conf.js (752 bytes)
CREATE angular-project/e2e/tsconfig.e2e.json (213 bytes)
CREATE angular-project/e2e/src/app.e2e-spec.ts (644 bytes)
CREATE angular-project/e2e/src/app.po.ts (251 bytes)
```

Рисунок 4.4 – Структура нового проекту

Новий проект містить усі необхідні конфігураційні файли та кореневий модуль програми. Після створення конфігурації проекту будуть встановлені усі пакети, що вказані у файлі package.json.

Нижче представлений список основних пакетів, що використовуються у застосунку Angular:

- @angular/core – найбільш важливий пакет, що містить основу Angular і його найбільш загальні елементи, такі як директиви і компоненти. Цей модуль завжди слід імпортувати в проект;
- @angular/common – цей пакет містить визначення всіх директив, служб і фільтрів, вбудованих в Angular, разом з відповідними класами;
- @angular/compiler – відповідає за компіляцію HTML-шаблонів в код, який дозволяє відобразити призначений для користувача інтерфейс програми;
- @angular/platform-browser – містить класи та функції, необхідні для компонування і взаємодії з моделлю DOM в контексті веб-браузера. Цей модуль дає можливість обробляти такі звичайні дії, як оновлення заголовка сторінки і налаштування розпізнавання жестів. Цей пакет містить також функції, необхідні для компіляції шаблонів в автономному режимі при виробничій експлуатації;
- @angular/platform-browser-dynamic – надає функцію завантаження, необхідну для ініціалізації додатків під час розробки;
- @angular/http – це HTTP-клієнт Angular, який дозволяє виконувати асинхронні запити до сервера;
- @angular/router – це вбудований в Angular маршрутизатор, що відповідає за навігацію застосунку.

4.2 Реалізація маршрутизації застосунку

Клієнтський застосунок має два основні модуля: модуль авторизації та модуль особистого кабінету користувача. Відповідно у системі повинні бути передбачені два маршрути: «/app», що відповідає за кабінет користувача та «/entrance», що відповідає за сторінки авторизації.

Фреймворк Angular дозволяє використовувати один маршрут з пустим шляхом, який буде використовуватися як маршрут за замовчуванням. Тому для модуля авторизації можна використати пустий шлях «/», що дозволить скоротити маршрут для сторінок.

Отже, по шляху «/» програма буде відображати модуль авторизації, а по шляху «/app» – модуль особистого кабінету користувача. Будь-який інший шлях буде автоматично перенаправлятися на сторінку авторизації.

Але сторінка модуля особистого кабінету повинна бути доступна лише для авторизованих користувачів, томи необхідно заборонити перехід неавторизованих користувачів на сторінку особистого кабінету та перенаправляти їх на сторінку входу до кабінету.

Для реалізації такого процесу Angular містить інструмент захисту шляхів маршрутизації [4].

Для захисту маршрутів використовується клас, що реалізує інтерфейс CanActivate, який містить метод canActivate. При переході на маршрут, для якого визначено клас для захисту маршруту, Angular викличе цей метод. Якщо результат виконання буде true, Angular активує цей маршрут та відобразить компонент, що відповідає даному маршруту.

5. МЕТОДИКА РОБОТИ КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Розроблена програмна система являється веб-застосунком, який не потрібно встановлювати на комп’ютер, а достатньо відкрити у браузері, маючи інтернет підключення. Застосунок підтримує останні версії усіх сучасних браузерів, а саме Google Chrome, Mozilla Firefox, Apple Safari ta Microsoft Edge.

При відкритті клієнтського додатку для неавторизованих користувачів відкривається сторінка авторизації з відповідною формою. Користувач повинен ввести свій логін, який являється його електронна адресу, яку він зазначив при реєстрації та пароль.

Якщо користувач ще не має облікового запису у системі, він може створити його перейшовши по посиланню під формою.

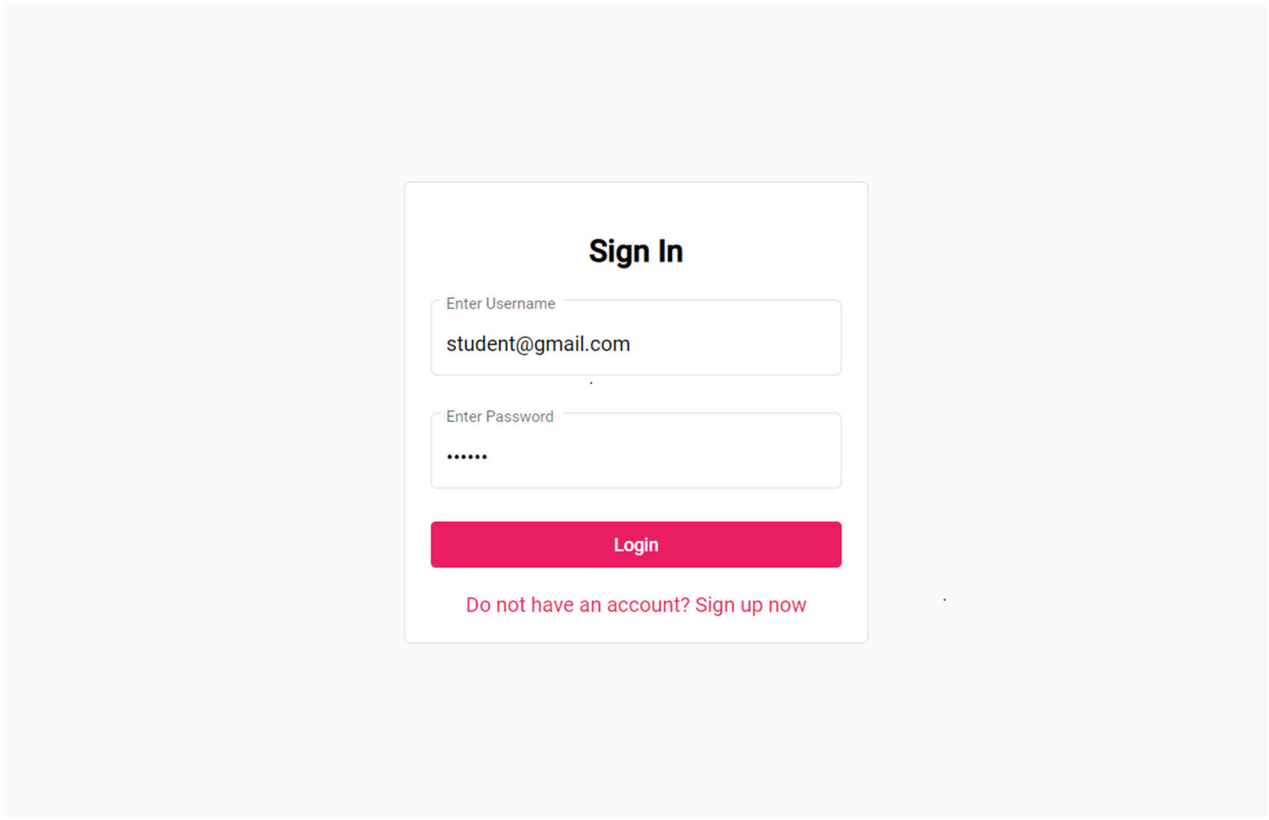

Рисунок 5.1 – Форма входу до кабінету

Форма реєстрації складається з чотирьох полів – ім’я користувача, адреса електронної пошти, телефон на пароль.

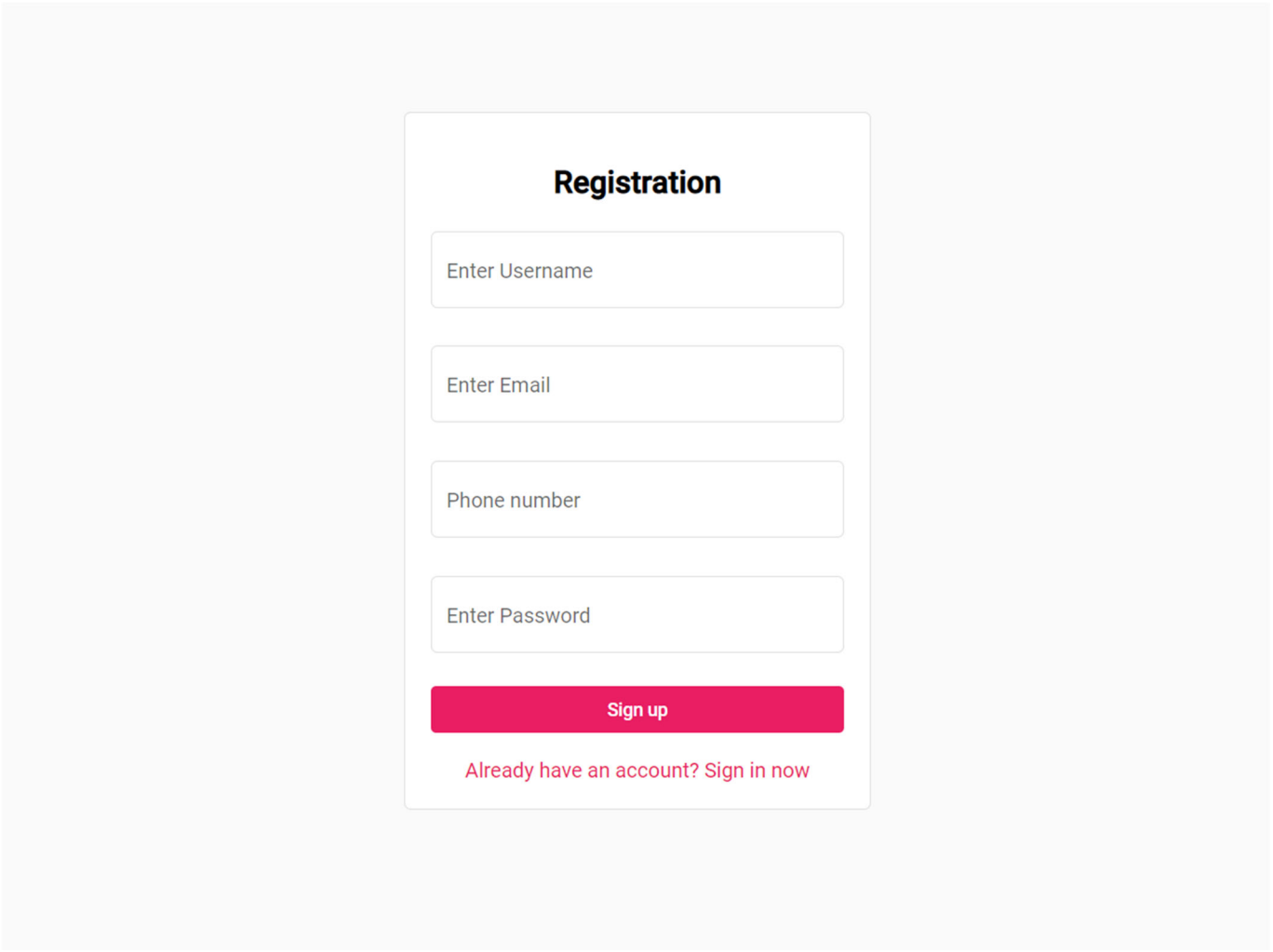

Рисунок 5.2 – Форма реєстрації

Всі дані, що вводяться до форми перевіряються перед відправкою, щоб уникнути створення користувача з незаповненими даними або з даними неправильного формату.

Після успішного створення облікового запису або авторизації користувач перенаправляється до особистого кабінету. Особистий кабінет має два підрозділи: поточна лекція та предмети.

У підрозділі поточної лекції відображаються інформація про лекцію, яка йде у даний момент. На екран виводиться список питань, що були задані під час лекції, а

при кліку на питання відкривається список відповідей, що були дані на це питання іншими студентами або викладачем.

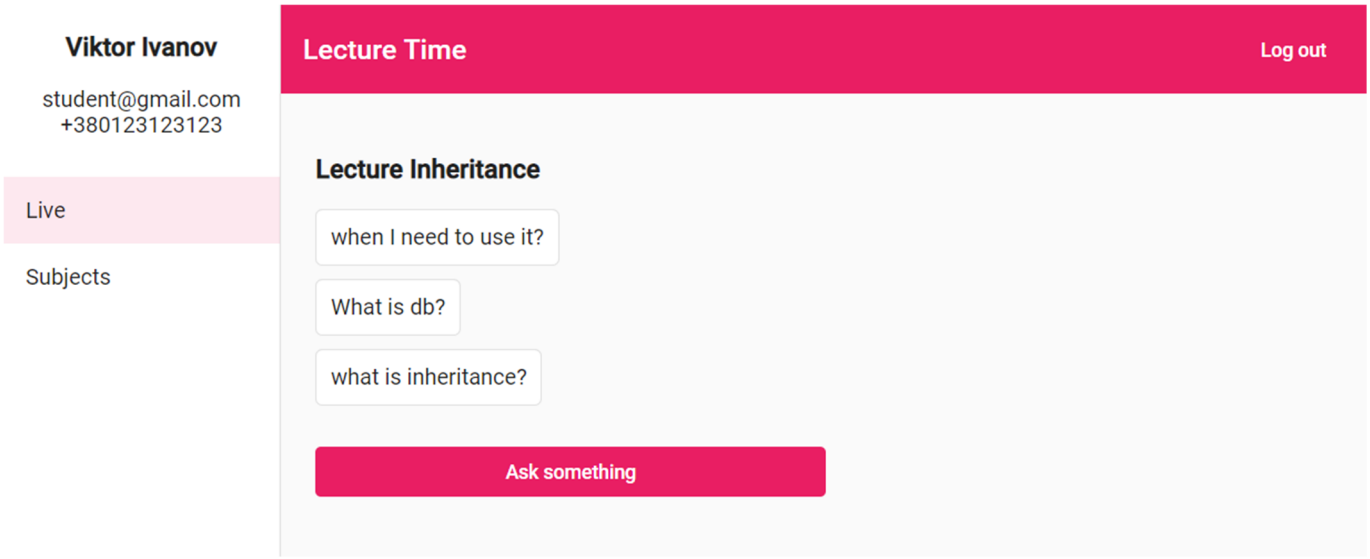

Рисунок 5.3 – Сторінка поточної лекції

Студент може задати своє питання натиснувши відповідну кнопку під списком питань. Після кліку на кнопку відкриється діалогове вікно з формою, у якій студент має ввести своє питання. Після підтвердження створення нового питання форма закривається, а питання з’являється у списку питань.

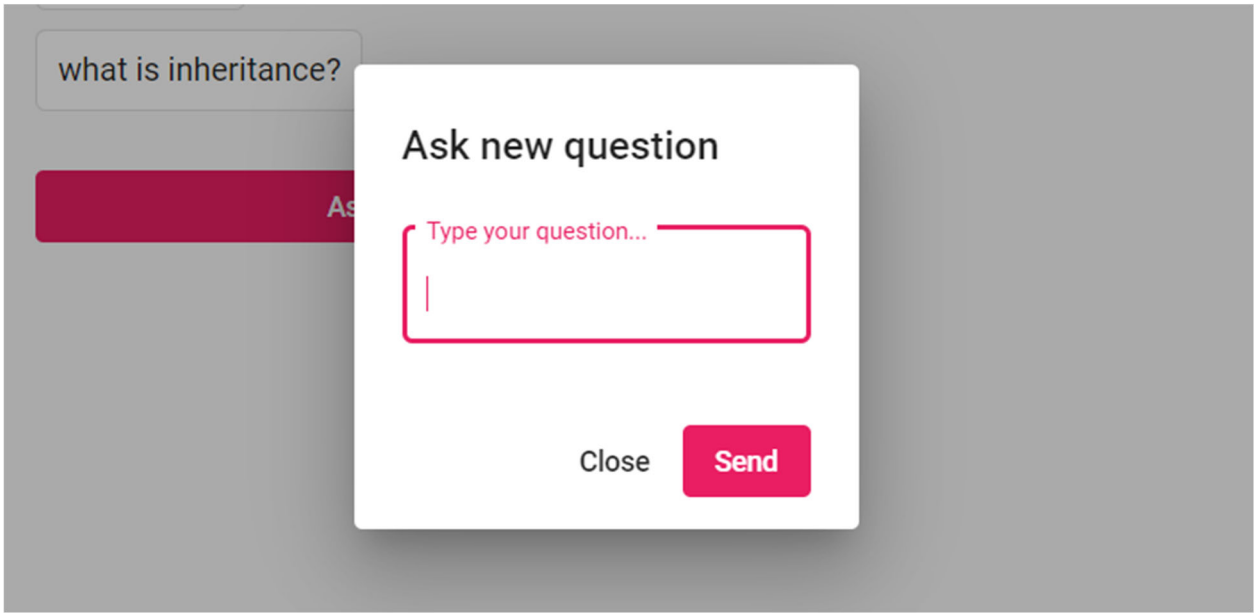

Рисунок 5.4 – Форма нового питання

На сторінці «Предмети» відображається список усіх предметів, що доступні користувачу.

Користувач може обрати будь який предмет, щоб подивитися лекції, що пройшли по цьому предмету.

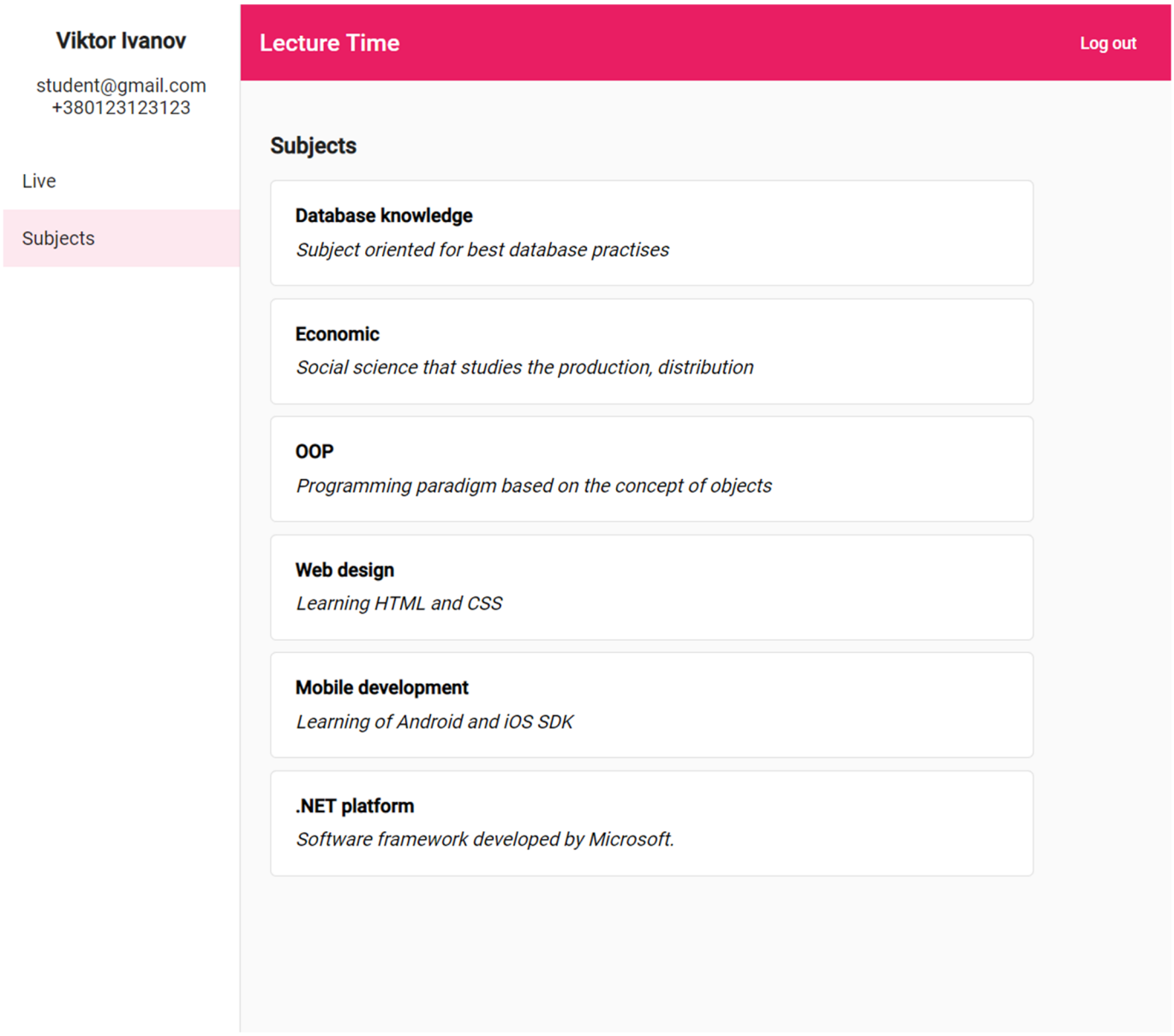

Рисунок 5.5 – Сторінка предметів

Коли користувач обрав предмет зі списку, йому відображається список минулих лекцій по цьому предмету.

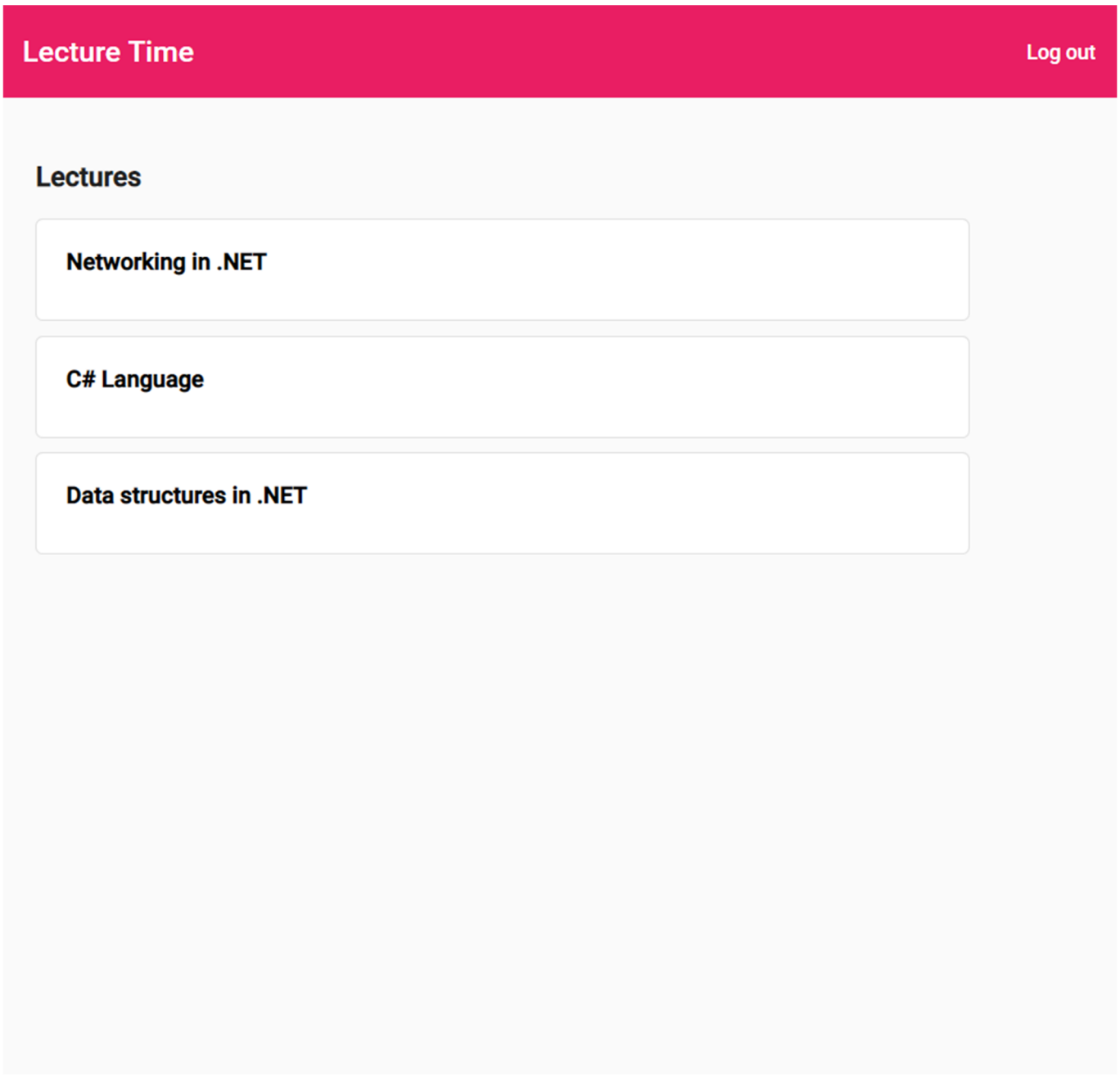

Рисунок 5.6 – Список лекцій

Обравши лекцію, користувач може подивитися список питань, що були задані під час лекції та відповіді, що були дані на ці питання.

Ця сторінка дещо відрізняється для користувачів групи “Викладач” та “Студент”. Це обумовлено тим, що студенти можуть тільки переглядати відповіді дані на поставлене запитання. Викладач може перевіряти питання та ставити «✔» напроти відповідей, які він вважає правильним.

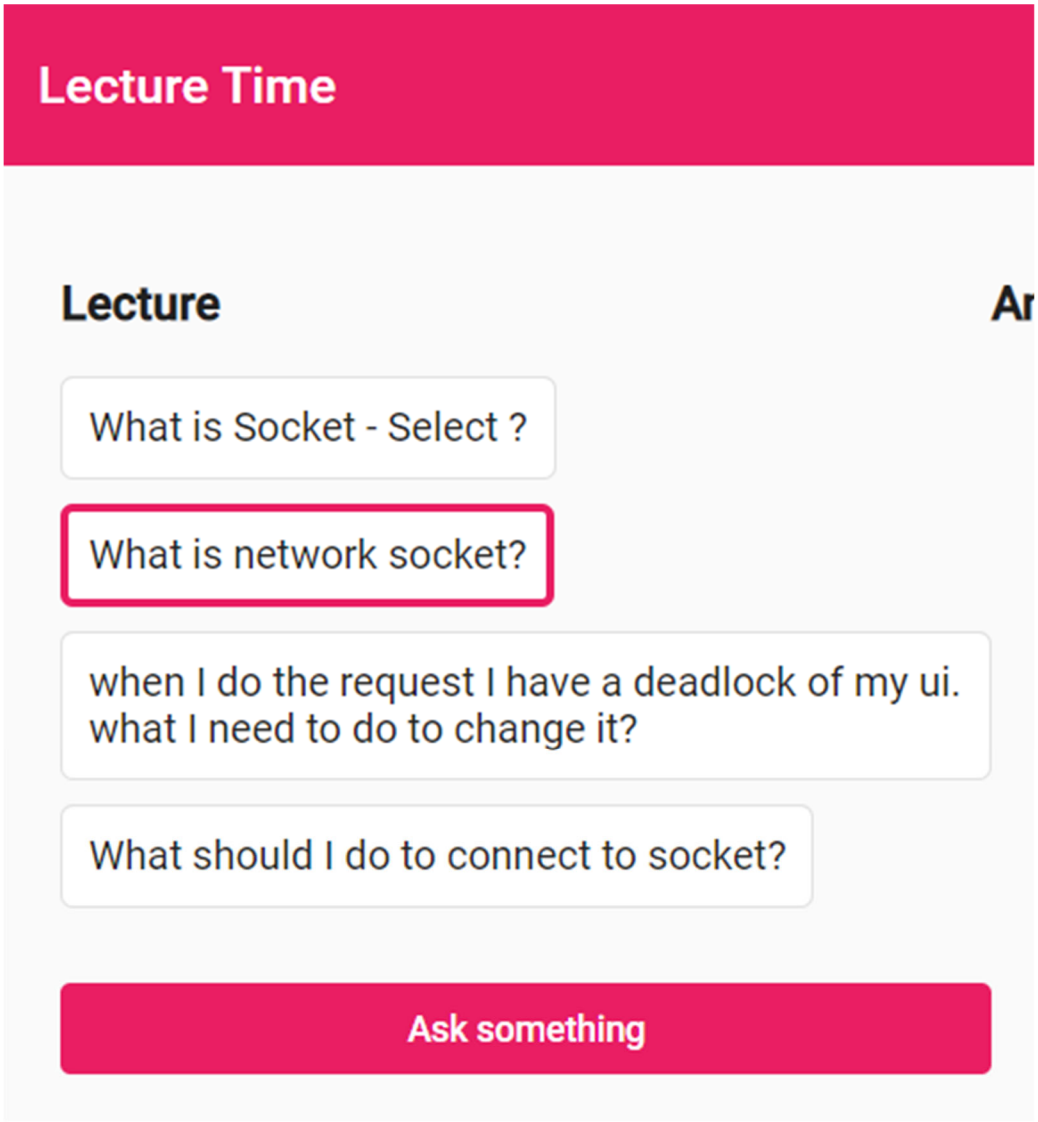

Рисунок 5.7 – Екран питань до лекції

В майбутньому планується реалізувати функціонал для змінення відповіді для користувачів групи “Викладач”. Тому викладачу не потрібно буде передруковувати часткову відповідь на питання з додаванням власних думок, а йому лише буде необхідно відредагувати близьку до правильної відповідь.

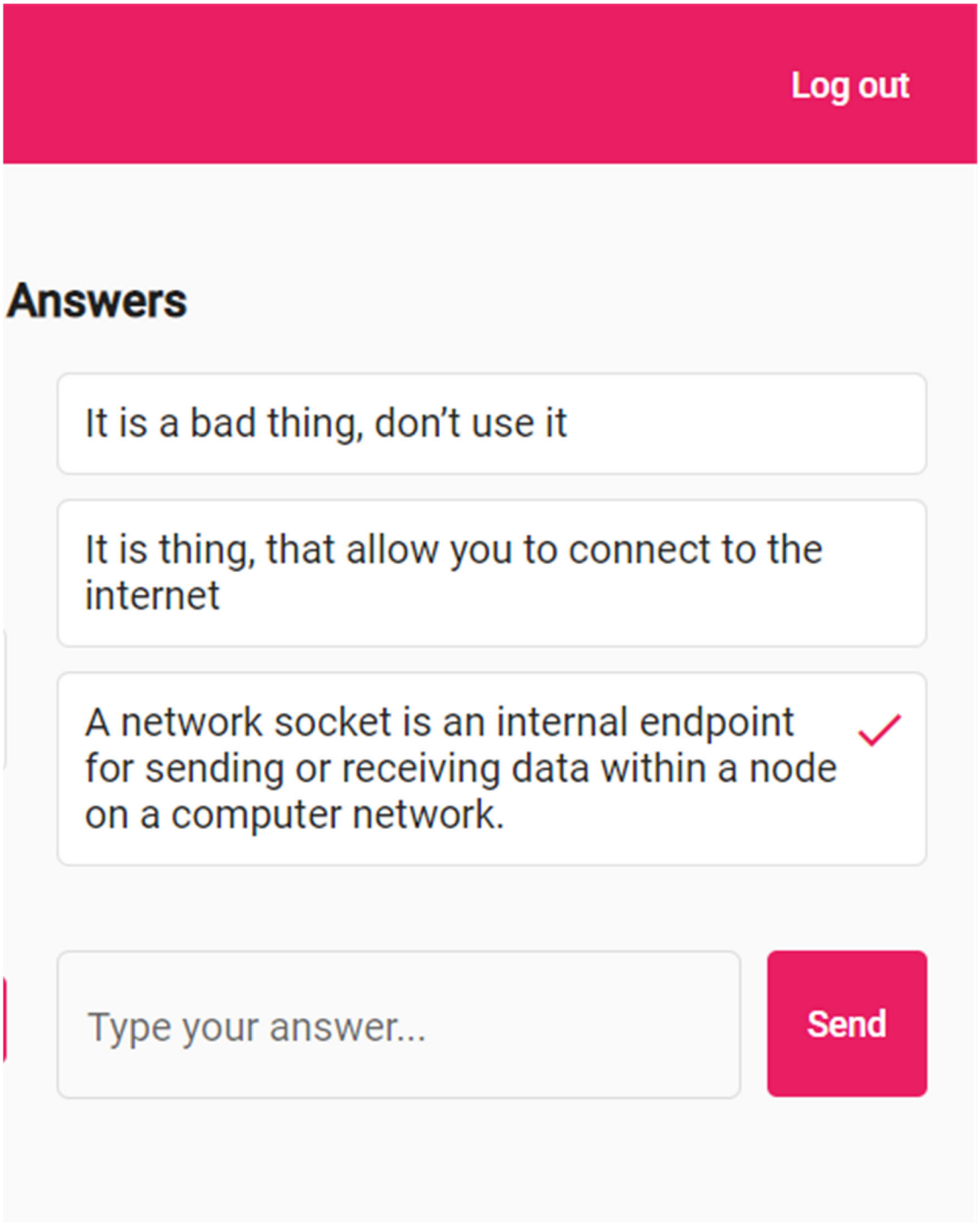

Рисунок 5.8 – Екран відповідей на питання до лекції

Якщо користувач хоче відповісти на питання, він може ввести питання у поле під вже існуючими відповідями та натиснути кнопку «Відправити».

ВИСНОВКИ

В ході проходження практики було розроблено програний продукт для поліпшення проведення занять, що вирішує проблему комунікації між викладачами та студентами під час занять.

Було проаналізовано існуючі програмні рішення схожої направленості та набуті навички практичного використання засобів розробки.

Було реалізовано клієнтський додаток, що працює на основі веб-технологій, до якого можна отримати доступ з пристрою, що має доступ до мережі інтернет через браузер.

Для розробки веб-додатку було використано мову TypeScript, основною перевагою якої є строга типізація, що допомагає уникнути помилок на етапі компіляції, та фреймворку для розробки односторінкових застосунків Angular.

У ролі бази даних та серверу було обрано сервіс хмарового сховища даних Firebase, що надає сервер як послугу і надає інструменти доступу до даних з клієнтського застосунку через API.

У системі реалізовано доступ до предметів та лекцій і дозволяє користувачам задавати питання до них.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Урутина Т. М. Коммуникативное взаимодействие преподавателей со студентами как фактор успешности обучения / Т. М. Урутина – Молодой ученый, 2015. – №18. – 552 с.

2. Леонтьев А. А. Педагогическое общение. / А. А. Леонтьев – М.: Знание, 1979. – 92 с.

3. Пьюривал С. Основы разработки веб-приложений / С. Пьюривал. – СПб.: Питер, 2015. – 272 с.

4. Фримен А. Angular для профессионалов / А. Фримен. – СПб.: Питер, 2018. – 800 с.

5. Гамма Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. – СПб.: Питер, 2015. – 368 с.

6. Wilken J. Angular in Action / Jeremy Wilken. – Shelter Island, New York: Manning Publications, 2018. – 320 с.

7. Файн Я. Angular и TypeScript. Сайтостроение для профессионалов / Я. Файн, А. Моисеев – СПб.: Питер, 2018. – 464 с.

8. Никсон Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5. 4-е изд. / Р. Никсон – СПб.: Питер, 2016. – 768 с.

9. Закас Н. JavaScript для профессиональных веб-разработчиков / Н. Закас – СПб.: Питер, 2015. – 960 с.

10. Clow M. Angular 5 Projects: Learn to Build Single Page Web Applications Using 70+ Projects / M. Clow – Apress, 2018. – 465 с.

ДОДАТОК А

Система поліпшення проведення занять (Web-додаток)

Специфікація

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТІ51178

Аркушів 2

Київ – 2019

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТІ51178	Записка.docx	Текстова частина дипломної роботи
Компоненти		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТІ51178	index.html	Головна сторінка застосунку

ДОДАТОК Б

Система поліпшення проведення занять (Web-додаток)

Текст програмного модулю

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТІ51178

Аркушів 5

Київ – 2019

Точка входу програми

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.error(err));
```

Головний модуль програми

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import { firebaseConfig } from './app.constants';
import { AngularFirestoreModule } from '@angular/fire/firestore';
import { AngularFireModule } from '@angular/fire';
import { AngularFireAuthModule } from '@angular/fire/auth';
import { AngularFireStorageModule } from '@angular/fire/storage';
import { AngularFireFunctionsModule } from '@angular/fire/functions';
import { AngularFireDatabaseModule } from '@angular/fire/database';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    BrowserAnimationsModule,
    AngularFireModule.initializeApp(firebaseConfig),
    AngularFirestoreModule,
    AngularFireAuthModule,
    AngularFireStorageModule,
    AngularFireFunctionsModule,
    AngularFireDatabaseModule,
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {
}
```

Головний маршрутизатор програми

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';

const routes: Routes = [
  {
    path: '',
    loadChildren: 'src/app/modules/entrance/entrance.module#EntranceModule',
  },
  {
    path: 'app',
    loadChildren: 'src/app/modules/base-navigation/base-navigation.module#BaseNavigationModule',
  },
  {
    path: '**',
    redirectTo: '/'
  }
];
```

```
@NgModule({
  imports: [RouterModule.forRoot(routes, {
    useHash: true,
  })],
  exports: [RouterModule]
})
export class AppRoutingModule {
}
```

Головний компонент програми

```
import { Component } from '@angular/core';
import { StorageService } from './servises/storage.service';
import { UsersService } from './servises/users.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent {

  constructor(private storage: StorageService, private users: UsersService) {
    this.storage.load();
    this.users.loadUser().subscribe();
  }
}
```

Сервіс авторизації

```
import { Injectable } from '@angular/core';
import { from } from 'rxjs';
import { AngularFireAuth } from '@angular/fire/auth';
import { AngularFirestore } from '@angular/fire/firestore';

@Injectable({
  providedIn: 'root'
})
export class AuthorizationService {

  constructor(
    private afAuth: AngularFireAuth,
    private afs: AngularFirestore,
  ) {
  }

  login(email, password) {
    return from(this.afAuth.auth.signInWithEmailAndPassword(email, password));
  }

  register(email, password) {
    return from(this.afAuth.auth.createUserWithEmailAndPassword(email, password));
  }

  signOut() {
    return from(this.afAuth.auth.signOut().then());
  }
}
```

Сервіс поточної лекції

```
import { Injectable } from '@angular/core';
import { BehaviorSubject } from 'rxjs';
import { AngularFireDatabase } from '@angular/fire/database';
import { AngularFirestore } from '@angular/fire/firestore';
import { StorageService } from './storage.service';
import { map, switchMap, tap } from 'rxjs/operators';
import { Live } from '../models/live';
import { Question } from '../models/question';
import { Answer } from '../models/answer';

@Injectable({
```

```
  providedIn: 'root'
})
export class LiveService {

  private questionsSubject: BehaviorSubject<Question[]> = new BehaviorSubject(null);
  private answersSubject: BehaviorSubject<Answer[]> = new BehaviorSubject(null);
  questions$ = this.questionsSubject.asObservable();
  answers$ = this.answersSubject.asObservable();
  selectedQuestion;
  lecture: Live;

  constructor(private store: AngularFirestore, private db: AngularFireDatabase, private storage: StorageService) {
  }

  loadQuestions() {
    return this.storage.live.valueChanges().pipe(
      tap(data => this.lecture = data[0]),
      switchMap(() => this.storage.questions.valueChanges()),
      map(data => (data as any[]).filter(q => q.lectureId === this.lecture.lecture)),
    ).subscribe(data => {
      this.questionsSubject.next(data);
    });
  }

  loadAnswers(questionId) {
    return this.storage.answers.valueChanges().pipe(
      map(data => (data as any[]).filter(a => a.questionId === questionId)),
    ).subscribe(data => {
      this.answersSubject.next(data);
    });
  }

  setSelectedQuestion(q) {
    this.selectedQuestion = q;
    return this.loadAnswers(q.id);
  }

  sendAnswer(text) {
    const answer = {
      id: this.selectedQuestion.id + this.answersSubject.value.length.toString(),
      questionId: this.selectedQuestion.id,
      text,
      accepted: false,
    };

    this.storage.answers.set(this.selectedQuestion.id + this.answersSubject.value.length.toString(), answer);
  }

  sendQuestion(text) {
    const question = {
      id: this.lecture.lecture + this.questionsSubject.value.length.toString(),
      text,
      lectureId: this.lecture.lecture,
    };
    this.storage.questions.set(this.lecture.lecture + this.questionsSubject.value.length.toString(), question);
  }

  toggle(answer: Answer) {
    answer.accepted = !answer.accepted;
    this.storage.answers.set(answer.id, answer);
  }
}
```

Сервіс підключення до серверу

```
import { Injectable } from '@angular/core';
import { AngularFireDatabase, AngularFireList } from '@angular/fire/database';
import { User } from '../models/user';
import { Question } from '../models/question';
import { Lecture } from '../models/lecture';
import { Subject } from '../models/subject';
import { Answer } from '../models/answer';
import { Live } from '../models/live';

@Injectable({
  providedIn: 'root'
})
export class StorageService {
```

```
  group = 'ti-51';

  questions: AngularFireList<Question>;
  lectures: AngularFireList<Lecture>;
  subjects: AngularFireList<Subject>;
  answers: AngularFireList<Answer>;
  live: AngularFireList<Live>;
  users: AngularFireList<User>;

  constructor(private db: AngularFireDatabase) {
  }

  load() {
    this.questions = this.db.list(`groups/${this.group}/questions`);
    this.lectures = this.db.list(`groups/${this.group}/lectures`);
    this.subjects = this.db.list(`groups/${this.group}/subjects`);
    this.answers = this.db.list(`groups/${this.group}/answers`);
    this.live = this.db.list(`groups/${this.group}/live`);
    this.users = this.db.list('users');
  }

}
```

ДОДАТОК В

Система поліпшення проведення занять (Web-додаток)

Опис програмного модулю

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТІ51178

Аркушів 5

Київ – 2019

АНОТАЦІЯ

Даний додаток містить опис програмної системи поліпшення проведення занять. Даний програмний продукт являє собою веб-додаток, який надає наступну функціональність:

- реєстрація та авторизація користувачів;
- доступ до поточної лекції;
- створення нового питання та відповіді до лекції;
- доступ до усіх предметів;
- доступ до усіх лекцій;

При розробці використовувалася мова TypeScript та фреймворк для розробки односторінкових веб-додатків у наступному середовищі: Microsoft Windows та WebStorm.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

Програмний продукт призначений для вирішення проблеми комунікації між викладачами та студентами. Программа являє собою односторінковий веб-додаток, доступ до якого можна отримати через веб-браузер з пристрою, що має доступ до мережі Інтернет.

Веб-додаток розроблений за допомогою мови програмування TypeScript та фреймворку для створення односторінкових клієнтських веб-застосунків Angular та мови розмітки HTML.

Розробка велася у наступному середовищі: операційній системі Microsoft Windows та інтегрованому середовищі WebStorm.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Розроблена система призначена вирішити проблему комунікації викладачів та студентів. Програмний продукт надає інтерфейс для доступу до особистого кабінету системи поліпшення проведення занять.

Користувач може отримати доступ до поточної лекції, задати питання до лекції та продивитися вже існуючі питання. Можна продивитися список відповідей до питання та ввести свою відповідь.

Також користувач може продивитися усі доступні для нього предметів та список минулих лекції до них.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

При відкритті додатку користувачу виводиться форма авторизації. Користувач може увійти до системи, використовуючи свої логін та пароль, або створити новий обліковий запис.

Після успішної авторизації користувач перенаправляється до сторінки поточної лекції, де відображається список питань до цієї лекції. Під списком питань виводиться форма вводу нового питання.

Користувач може обрати питання, натиснувши ліву клавішу миші на ньому, після чого відкриється список відповідей до нього.

Користувач може ввести нову відповідь на питання, після чого вона з’явиться у списку.

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Розробка програмного продукту велася у середовищі операційної системи Microsoft Windows та інтегрованого середовища розробки WebStorm, який надає зручні інструменти розробки мовою TypeScript та має підтримку фреймворку Angular.

Для запуску програмного продукту було використано браузер Google Chrome, який підтримує усі нові стандарти та має зручні інструменти налагодження.

Доступ до додатку можна отримати з будь-якого браузера з пристрою, що має доступ до мережі інтернет.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Так як даний програмний продукт являє собою веб-додаток, скомпільовані файли програми потрібно розвернути на веб-сервері.

Після того, як веб-додаток користувачам не потрібно інсталювати програму, достатньо лише перейти за адресую сервера у веб-браузері.